ADAPTIVE INVERSE CONTROL
OF PLANTS WITH DISTURBANCES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Gregory L. Plett

May 1998

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Bernard Widrow
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Gene F. Franklin

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Fouad A. Tobagi

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

The goal of control theory is to make a given dynamical system (the “plant”) behave in a user-specified manner as accurately as possible. This objective may be broken down into three separate tasks: stabilization of the plant dynamics; control of plant dynamics; and control of plant disturbance. Conventionally, one uses feedback to treat all three problems simultaneously. Compromises are necessary to achieve good solutions.

Adaptive inverse control is a method to treat the three control tasks separately. First, the plant is stabilized; secondly, the plant is controlled using a feedforward controller; thirdly, a disturbance canceller is used to reject plant disturbances. Adaptive filters are used as controller and disturbance canceller, and algorithms adapt the transfer functions of the filters to achieve excellent control.

Prior work in adaptive inverse control has focused mainly on feedforward control and disturbance cancelling for single-input single-output linear plants, and on feedforward control for single-input single-output nonlinear plants. This dissertation extends the prior work to encompass feedforward control and disturbance cancelling for single-input single-output or multi-input multi-output, linear or nonlinear plants.

An important part of this work is the development of a gradient-descent based algorithm for updating the weights of either the controller or the disturbance cancelling filters. The algorithm decouples nicely, allowing separate implementation of the adaptive controller, plant model and disturbance canceller; only local information is needed for the weight update. Very general user-specified constraints on the control effort may be satisfied, and excellent disturbance rejection can be achieved. Additionally, it is shown how to compensate for the effects of non-ideal sensors.

The added functionality does not come at the expense of algorithmic or structural complexity. The final control architecture in this dissertation is much simpler than any previously reported. Simulation results are presented to verify the analysis and synthesis methods. Overall, excellent results are obtained.

Acknowledgments

Plans fail for lack of counsel,
but with many advisers they succeed.
—Proverbs 15:22

הָפֵר מַחֲשָׁבוֹת בְּאֵין סוֹד
וּבְרֹב יוֹעֲצִים תָּקוּם
—משלי טו22

I would like to express my gratitude to my principle advisor Dr. Bernard Widrow. I am deeply indebted to him for many reasons: he has provided encouragement and technical support; he has opened up many academic opportunities of which I would not have otherwise dreamed; and he has enthusiastically championed my cause.

I would also like to thank Dr. Gene Franklin for generously coming out of retirement to serve on my reading committee. In addition, I am grateful to Dr. Fouad Tobagi for being a reader. I have worked with Dr. Tobagi on a number of occasions, and I have been encouraged to greater and greater professionalism by his unwavering example.

Throughout my life, I have been influenced by many rôle-models. I would like to especially acknowledge my parents. They have encouraged me to pursue my interests, and have been both accepting and supportive of my decisions. They gracefully bore the horror of watching their first-born go off on his own to this, the opposite side of the continent. I am indebted to them for many things, but most importantly for their example of what it means to be a person of faith. Dad, Mom, Jonathan: I love you very much.

My engineering history began at Carleton University. This excellent school prepared me very well for my subsequent studies here at Stanford. The camaraderie and friendly competition among fellow students played an important rôle in stimulating interest in my studies then, and in the possibility of an academic career. A special thanks to Dr. Jeff Carruthers who has been a constant friend and who was instrumental in my decision to study at Stanford.

While sojourning here, I have met many interesting people and have made some good friends. Joice DeBolt makes things run smoothly around here, and quietly tolerates my frequent interruptions. Without her, nothing would get done by this group. Thanks also to all the past and present members of “the Zoo” including: Dr. Françoise Beaufays, Dr. Michel Bilello, Daniel Carbonell, Takeshi “Keish” Doi, Dr. Mike Lehr, Dr. Ming-Chang Liu, the elusive James McNames, Chang-Yun Seong and Dr. Raymond Shen. Raymond deserves a special thanks for patiently suffering through many technical discussions with me, but also for the times of Bible study, and for being someone to talk with about deeper issues. Raymond: I consider you a friend for life.

Of all the people I have met at Stanford, my greatest friendship has been forged with the woman who is now my wife, Linda-Rocio Garcia Plett. Linda has become my best friend and confidante, and has provided more love, respect, encouragement and acceptance than I could have ever hoped for. I am a very fortunate man. Linda: I love you with all my heart.

Although many people have influenced the person who I am, by far my biggest debt of gratitude is to my God and Savior Jesus Christ. Through his word, the Bible, I am learning how to become a man of faith and a man of character. As part of this life-journey, he brought me to Stanford and has helped me through some difficult times. He has also brought me many, many joys, and I look forward to see how he shapes the future. I dedicate this work to him.

Oh, the depth of the riches of the wisdom and knowledge of God!
How unsearchable his judgments, and his paths beyond tracing out!
“Who has known the mind of the Lord?
Or who has been his counselor?”
“Who has ever given to God, that God should repay him?”
For from him and through him and to him are all things.
To him be the glory forever! Amen.
—Romans 11:33–36.

A “1967 scholarship” from the Natural Sciences and Engineering Research Council of Canada supported me for four years. Combined with the “Anderson Fellowship,” it allowed me to study at this, the premier university in the world (Jeff may disagree ☺). This work was also supported in part by the following organizations: the U. S. Army Belvoir R D & E Center under contract DAAK70–92–K–0003; the National Science Foundation under contract ECS–9522085; and the Electric Power Research Institute under contract WO8016–17. Any opinions, findings, or conclusions contained herein are those of the author, and do not necessarily reflect the views of the sponsoring agencies.

Contents

Abstract **v**

Acknowledgments **vii**

1 Adaptive Inverse Control **1**

1.1 Introduction 1

1.2 Discrete-Time Control Systems 3

1.3 The Framework of Adaptive Inverse Control 4

1.4 Author’s Contributions 6

1.5 Outline 8

2 Adaptive (Linear and Nonlinear) Digital Filters **9**

2.1 Introduction 9

2.2 Linear Adaptive Filters 10

2.2.1 Structure of Linear Adaptive Filters 10

2.2.2 Adapting Linear Adaptive Filters 11

2.2.3 Optimal Solution for Linear Adaptive Filters 13

2.3 Nonlinear Adaptive Filters 14

2.3.1 Structure of Nonlinear Adaptive Filters 14

2.3.2 Adapting Nonlinear Adaptive Filters 16

2.3.3 Optimal Solution for Nonlinear Adaptive Filters 20

2.4 Example: Linear and Nonlinear System Identification 21

2.5 Summary 24

3 Plants Used as Examples 25
3.1 Introduction . . . 25
3.2 Linear Plants . . . 25
3.2.1 Linear SISO Plant . . . 26
3.2.2 Linear MIMO Plant . . . 30
3.3 Nonlinear Plants . . . 35
3.3.1 Nonlinear SISO Plant . . . 37
3.3.2 Nonlinear MIMO Plant . . . 40
3.4 Summary . . . 44

4 Constrained Adaptive Feedforward Control 45
4.1 Introduction . . . 45
4.2 Analysis of Constrained Linear Control . . . 46
4.2.1 A Working Linear MIMO Control Architecture . . . 46
4.2.2 Constraint on the Control Effort . . . 48
4.2.3 An Approximate Solution for the Constrained Controller . . . 49
4.2.4 Control Effort and Controlling Nonminimum-Phase Plants . . . 54
4.3 Synthesis of the Constrained Controller via the BPTM Algorithm . . . 57
4.3.1 Linear FIR Plant Model, Linear FIR Controller . . . 60
4.3.2 Nonlinear NARX Plant Model, Nonlinear NARX Controller . . . 63
4.3.3 Separability for Efficient Implementation . . . 65
4.3.4 Initialization of Linear Controllers . . . 66
4.3.5 Initialization of Nonlinear Controllers . . . 68
4.4 Simulation Examples . . . 69
4.4.1 Minimum-Phase Linear SISO Plant . . . 69
4.4.2 Nonminimum-Phase Linear SISO Plant . . . 71
4.4.3 Linear MIMO Plant . . . 74
4.4.4 Nonlinear SISO Plant . . . 78
4.4.5 Nonlinear MIMO Plant . . . 80
4.5 Summary . . . 83

5 Closing the Loop: Disturbance Cancelling 85
5.1 Introduction . . . 85
5.2 Analysis of Disturbance Cancelling . . . 86

5.2.1 Correctness of Feedforward Design in the Presence of Disturbance 86
5.2.2 Conventional Disturbance Rejection Methods Fail 87
5.2.3 A Solution Allowing On-Line Adaptation of $\widehat{P}$ 91
5.2.4 Structure of the Disturbance Canceller 94
5.3 Synthesis of the Disturbance Canceller via the BPTM Algorithm 97
5.3.1 Controller Feedback 97
5.3.2 Estimator Plus Controller Feedback 98
5.3.3 Training X In-Place 99
5.4 Simulation Examples 101
5.4.1 Minimum-Phase Linear SISO Plant 101
5.4.2 Nonminimum-Phase Linear SISO Plant 107
5.4.3 Linear MIMO Plant 109
5.4.4 Nonlinear Plants 110
5.5 Summary 111

6 Imperfect Sensors **115**
6.1 Introduction 115
6.2 Analysis and Synthesis of Adaptive Inverse Control with an Imperfect Sensor . . . 116
6.2.1 Effect of S on Plant Modeling 116
6.2.2 Effect of S on Feedforward Control 117
6.2.3 Effect of S on Disturbance Cancelling 120
6.3 Simulation Examples 124
6.3.1 The Sensors 124
6.3.2 Linear Sensor; No Sensor Noise 126
6.3.3 Linear Sensor; White Sensor Noise 130
6.3.4 Linear Sensor; Correlated Sensor Noise 131
6.3.5 Nonlinear Sensor 131
6.4 Summary 131

7 Conclusions and Future Work **135**
7.1 Summary 135
7.1.1 Constrained Adaptive Feedforward Control 135
7.1.2 Disturbance Cancelling 136
7.1.3 Imperfect Sensors 137

7.2 Future Work 137
7.3 Final Comments 139

A Stability Analysis of the LMS and Backpropagation Algorithms 141
A.1 Introduction 141
A.2 Linear Filters and the LMS Algorithm 141
A.2.1 Stability of the LMS Algorithm 142
A.3 Neural Networks and the Backpropagation Algorithm 144
A.3.1 Stability of the Backpropagation Algorithm 147
A.4 Summary 149

B More Nonlinear SISO Examples 151
B.1 Introduction 151
B.2 System Identification 151
B.3 Feedforward Control 157
B.4 Disturbance Cancelling 168
B.5 Summary 170

Bibliography 171

List of Tables

4.1 Filter architectures used in this chapter 84

5.1 Steady-state mean squared system error for all plants simulated 112
5.2 Filter architectures used in this chapter 114

6.1 Transfer functions of the minimum-phase sensors 127
6.2 Steady-state mean squared system error for disturbance cancelling 129
6.3 Filter architectures used in this chapter 134

B.1 Filter architectures used in this appendix 168

List of Figures

1.1 Classical control system 3
1.2 Digital control system 4
1.3 Internal model control 5
1.4 Adaptive inverse control 6

2.1 Symbolic representation of an adaptive filter 9
2.2 Structure of a linear filter 10
2.3 Structure of a nonlinear filter 15
2.4 System identification 22
2.5 Nonlinear system identification 24

3.1 Tank temperature control 27
3.2 Discrete-time impulse responses and pole-zero plots for the linear SISO plants . . . 29
3.3 Aircraft yaw-rate and bank-angle control 31
3.4 Uncompensated jet impulse responses 32
3.5 Simple feedback system to increase the damping of one of the lateral modes 33
3.6 Compensated, discrete-time jet impulse responses 34
3.7 Simulating wind disturbances 36
3.8 Illustration of heading (yaw) angle, $\psi(t)$ 37
3.9 Block diagram of ship yaw dynamics from wheel angle δ_w to heading angle ψ . . . 38
3.10 Step responses of stabilized and unstabilized ship 39
3.11 Block diagram of stabilized ship 39
3.12 Simulating sea disturbances 41
3.13 Two-link robot arm 41

3.14 Stabilized robot 43

4.1 MIMO controller design 47
4.2 Step response of controlled minimum-phase tank 49
4.3 An algorithm for synthesizing a constrained linear MIMO controller 52
4.4 Comparing magnitude responses of theoretical and adapted controllers 53
4.5 Histograms of control effort 55
4.6 Two Shannon-Bode optimal controllers 56
4.7 Steady-state system MSE versus system delay 57
4.8 Conceptual block diagram of a system used to train the controller C 58
4.9 Structure diagram illustrating the BPTM method 58
4.10 Algorithm to adapt an FIR controller for an FIR plant model 63
4.11 Algorithm to adapt a NARX controller for a NARX plant model 65
4.12 Algorithmic independence of BPTM on the structures of C and $\widehat{P}$ 66
4.13 Learning curves for fast and slow learning 68
4.14 Diagram showing a possible method of initializing a nonlinear controller 69
4.15 Penalty function used on the magnitude of the control effort 70
4.16 Impulse responses of controllers for the minimum-phase tank 71
4.17 Tracking performance of three controllers for the minimum-phase tank example . . 72
4.18 Step response of three controllers for the minimum-phase tank example 73
4.19 Impulse response of the constrained controller for the nonminimum-phase tank . . 74
4.20 Impulse responses of constrained controllers for the nonminimum-phase tank . . . 75
4.21 The four impulse responses comprising the Boeing 747 controller 76
4.22 Tracking performance and control effort for the Boeing 747 controller 77
4.23 Histograms of slew rate for the Boeing 747 example 78
4.24 Tracking performance of ship controller 79
4.25 Steady-state system mean-squared error versus the system delay 80
4.26 Tracking performance and control effort for the robot controller 81
4.27 Tracking performance and control effort for the constrained robot controller 82
4.28 Histograms of slew rate of controllers for the robot example 83

5.1 Plant modeling in the context of feedforward control 86
5.2 Two methods to close the loop 88
5.3 An intermediate step when analyzing the convergence of $\widehat{P}$ 89

5.4 Correct on-line plant modeling and disturbance cancelling 92
5.5 Input-output timing of a discrete-time control system 94
5.6 A useful way of looking at the feedforward system dynamics 95
5.7 Internal structure of X 96
5.8 Internal structure of X if the plant is linear 96
5.9 Disturbance cancelling via controller feedback 98
5.10 Disturbance cancelling via estimator plus controller feedback 99
5.11 An integrated nonlinear MIMO system 101
5.12 Disturbance-cancelling for unconstrained minimum-phase linear SISO system . . . 102
5.13 Disturbance-cancelling for constrained minimum-phase linear SISO system 103
5.14 Biased and unbiased plant models 105
5.15 Various aspects of disturbance cancelling for the minimum-phase tank system . . . 106
5.16 Disturbance-cancelling for unconstrained nonminimum-phase linear SISO system . 107
5.17 Disturbance-cancelling for constrained nonminimum-phase linear SISO system . . 108
5.18 Disturbance cancelling filter X 109
5.19 Disturbance-cancelling for linear MIMO system 110

6.1 A non-ideal sensor 115
6.2 The system model is an adaptive model of the combined dynamics of S and P . . . 117
6.3 Uncompensated feedforward control 118
6.4 Compensated feedforward control 119
6.5 Generating F 122
6.6 Internal structure of X if there is sensor noise 123
6.7 Integrated, compensated control system 125
6.8 True sensor dynamics 126
6.9 Impulse responses of plant and system 127
6.10 Sensitivity of system error to sensor uncertainty 128
6.11 Tracking with uncertain sensor 129
6.12 Tracking with reference model and uncertain sensor 130
6.13 Tracking with nonlinear sensor 132

A.1 Finite-impulse-response (FIR) linear filter 142
A.2 Artificial neuron 145
A.3 Nonlinear transversal filter 145

B.1 Block diagram of system 1 152
B.2 Block diagram of system 2 153
B.3 Block diagram of system 3 154
B.4 Block diagram of system 4 154
B.5 Block diagram of system 5 155
B.6 Block diagram of system 6 156
B.7 Block diagram of system 7 156
B.8 System identification in the absense of disturbance 158
B.9 System identification in the presence of disturbance 159
B.10 Logarithm of mean-squared system error plotted versus control system delay . . . 160
B.11 Feedforward control of system 1 161
B.12 Feedforward control of system 2 162
B.13 Feedforward control of system 3 163
B.14 Feedforward control of system 4 164
B.15 Feedforward control of system 5 165
B.16 Feedforward control of system 6 166
B.17 Feedforward control of system 7 167
B.18 Plots showing disturbance cancelation 169

List of Notation

Mathematical Operators

$\mathbb{E}\left[\cdot\right]$ Returns the expected value of $(\cdot)$.

$\left[\cdot\right]_+$ Takes the inverse z-transform of $(\cdot)$, sets the non-causal part to zero, and returns the z-transform of the remaining causal part.

$\left(\cdot\right)^+(z)$ The minimum-phase part of the spectral factorization of $\left(\cdot\right)(z)$, such that $\left(\cdot\right)(z) = \left(\cdot\right)^+(z)\,\left(\cdot\right)^-(z)$.

$\left(\cdot\right)^-(z)$ The nonminimum-phase part of the spectral factorization of $\left(\cdot\right)(z)$, such that $\left(\cdot\right)(z) = \left(\cdot\right)^+(z)\,\left(\cdot\right)^-(z)$.

$\mathcal{Z}\left\{\cdot\right\}$ Returns the z-transform. When $(\cdot)$ is a Laplace transform, this operator takes the inverse Laplace transform of its operand, samples the result, and returns the z-transform of the sampled signal.

$\partial A/\partial B$ Returns the ordinary partial derivative of A with respect to B.

$\partial^+ A/\partial B$ Returns the ordered partial derivative of A with respect to B. The ordered derivative is equal to the total derivative for ordered systems. An ordered system is a mathematical system of equations which is evaluated in a specific sequence.

Adaptive Filter Architectures

$\mathrm{FIR}_{\mathrm{(a,0):b}}$ An FIR filter with a tapped delays on each input, and b outputs.

$\mathrm{IIR}_{\mathrm{(a,b):c}}$ An IIR filter with a tapped delays on each exogeneous input, b tapped delays on each feedback input, and c outputs.

$\mathcal{N}_{\mathrm{(a,b):c:d...}}$ A neural-network based NARX filter with one stream of exogeneous input. There are a tapped delays on each input, b tapped delays on each output; c neurons in the first layer, d neurons in the second layer, and so forth.

$\mathcal{N}_{([a_1,a_2],b):c:d\ldots}$ A neural-network based NARX filter with two streams of exogeneous input. There are a_1 tapped delays on each input from the first stream, a_2 tapped delays on each input from the second stream, b tapped delays on each output; c neurons in the first layer, d neurons in the second layer, and so forth.

General Acronyms

ARMA AutoRegressive Moving Average: An infinite impulse response filter.

FIR Finite impulse response.

i.i.d. Independent and identically distributed (random variables).

IIR Infinite impulse response.

MSE Mean Squared Error.

NARX Nonlinear AutoRegressive filter with eXogeneous input.

Algorithm Names and Acronyms

Backprop Adapts feedforward neural networks (also known as “Backpropagation”).

BPTT BackPropagation Through Time: Adapts recurrent neural networks.

BPTM BackPropagation Through (Plant) Model: Algorithm developed in this dissertation to adapt the controller and disturbance canceller.

LMS Least Mean Square: Adapts FIR linear filters.

RTRL Real Time Recurrent Learning: Adapts recurrent neural networks in real time.

Symbols for Adaptive Elements

C The adaptive controller.

C_{COPY} A filter whose weights are a digital copy of those in the adaptive controller C.

E An adaptive estimator used to predict future disturbance values.

E_{COPY} A filter whose weights are a digital copy of those in the adaptive estimator E.

F A filter used to predict the current sensor noise given the past values of estimated disturbance.

M The reference model.

P The plant.

$\widehat{P}$ The adaptive plant model.

$\widehat{P}_{\text{COPY}}$ A filter whose weights are a digital copy of those in the adaptive plant model $\widehat{P}$.

S The dynamics of the sensor.

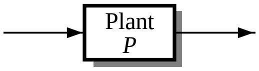
$\widehat{S}$	The model of the estimated sensor dynamics.
SP	The combined dynamics of the plant and sensor.
$\widehat{SP}$	The adaptive system model.
$\widehat{SP}_{\text{COPY}}$	A filter whose weights are a digital copy of those in the adaptive system model $\widehat{SP}$.
X	The adaptive disturbance canceller.

Common Signals

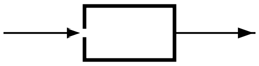
c_k	Impulse response of linear controller.
d_k	Desired response for system output.
$\vec{d}_k$	The infinite vector containing $d_k, d_{k-1} \ldots$
$\tilde{d}_k$	Desired response for sensor output.
$e_k^{(\text{mod})}$	Plant modeling error.
$e_k^{(\text{sys})}$	System error.
$\tilde{e}_k^{(\text{sys})}$	Measured system error.
$\tilde{\varepsilon}_k$	Modified system error.
p_k	Impulse response of linear plant model.
r_k	Reference input.
$\vec{r}_k$	The infinite vector containing $r_k, r_{k-1} \ldots$
u_k	Controller output.
$\vec{u}_k$	The infinite vector containing $u_k, u_{k-1} \ldots$
$\tilde{u}_k$	Disturbance canceller output.
$\vec{\tilde{u}}_k$	The infinite vector containing $\tilde{u}_k, \tilde{u}_{k-1} \ldots$
v_k	Sensor noise.
$\vec{v}_k$	The infinite vector containing $v_k, v_{k-1} \ldots$
$\hat{v}_k$	Estimate of sensor noise.
w_k	Disturbance at plant output.
$\vec{w}_k$	The infinite vector containing $w_k, w_{k-1} \ldots$
$\widehat{w}_k$	Estimate of disturbance at plant output.
$\vec{\widehat{w}}_k$	The infinite vector containing $\widehat{w}_k, \widehat{w}_{k-1} \ldots$
x_k	Impulse response of linear disturbance canceller.
y_k	Plant output (including disturbance).
$\vec{y}_k$	The infinite vector containing $y_k, y_{k-1} \ldots$
$\tilde{y}_k$	Measured plant output (including sensor dynamics and noise).

$\hat{y}_k$ Plant model output.

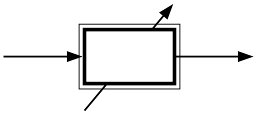
Block Diagram Elements



The plant.



Linear or nonlinear non-adaptive filter.



Linear or nonlinear adaptive filter.

Chapter 1

Adaptive Inverse Control

Everything should be made as simple as possible, but not simpler.
—Albert Einstein

1.1 Introduction

The goal of control theory is to make a given dynamical system (the "plant") behave in a user-specified manner as accurately and robustly as possible. Dynamical systems we might wish to control exist in great variety; but generally, they may be divided into several categories. First, any plant is said to be either linear or nonlinear. Linear dynamical systems obey the superposition principle and nonlinear ones do not [19]. Each plant is also either single-input-single-output (SISO), or multi-input-multi-output (MIMO).[1]

Control problems are also classified. For any plant, we may wish to address one (or more) of three basic control problems. These are:

- *Regulator control*, which is concerned largely with the transient and steady-state response of the plant in recovering from disturbances in a timely and robust fashion.
- *Servo (or tracking) control*, which is concerned with the transient and steady-state response of the plant in following a given trajectory closely, quickly and smoothly.
- *Terminal control*, which is concerned with the ability to move the system output from one state to another, *without concern for the trajectory*. Tradeoffs are made between optimality in terms of time or resource use, the magnitude of the control signal, and final-state accuracy.

[1] Multi-input-single-output (MISO) and single-input-multi-output (SIMO) systems also exist, but are treated here as special cases of MIMO systems.

In all three cases, constraints are typically imposed on the control signal and must be properly handled by the controller. Regulator and servo control problems are similar enough that they are usually considered together. The regulator problem is a degenerate case of the servo control problem where the goal is to track a constant input signal. Terminal control is sufficiently different to be treated separately. In this dissertation, only regulator and servo control problems are considered.

In classical and modern analog control systems, precise regulator and servo control is accomplished with compensation networks and feedback [7]. Likewise, conventional discrete time control systems build on this classical material and use similar means to control a plant using a digital computer [8].

These methods work well when the plant is linear and its dynamics are well understood. However, plant dynamics are rarely known with precision and may be too nonlinear to control with a linear controller. Current research in post-modern (!) control design [15, 47] searches for robust stability and performance for linear systems whose dynamics are not completely understood. There is much work yet to be done in this field, and indeed it seems that the need to address the control of nonlinear systems has hardly been touched. As one researcher states: “From a mathematical point of view, even the control of known nonlinear dynamical systems is a formidable problem. This becomes substantially more complex when the representation of the system is not completely known [22].”

In this dissertation, we address the control of linear, nonlinear, SISO and MIMO systems with adaptive controllers—“a controller with adjustable parameters and a mechanism for adjusting the parameters [1, p. 1].” The field of adaptive control, unlike the disciplines of classical and state-space controller design, is still much of an “art.” However, in principle, much is to be gained from using adaptive control techniques. Among these are:

- The possibility of controlling increasingly complicated dynamic systems,
- Incorporation of design constraints in a very practical fashion,
- Greater precision of control due to better plant modeling,
- Robustness to variation in internal plant parameters (process variations), and,
- Resilience to variation in the character of the disturbances.

In this dissertation, we investigate the control problem under the framework of *adaptive inverse control* [45]. Adaptive signal processing methods are used to control either linear or nonlinear, SISO or MIMO plants. As the control of SISO linear and undisturbed SISO nonlinear plants has already been extensively treated elsewhere [2, 4, 45], the focus here is on the control of disturbed MIMO and nonlinear plants. The solution is amazingly simple and stunningly powerful.

1.2 Discrete-Time Control Systems

The field of classical control theory concerns itself with the task of servo or regulator control of linear analog plants. A controller designed according to this discipline will be a linear analog device, and may be implemented with operational amplifiers, resistors, capacitors and the like. Figure 1.1 shows a schematic diagram of a classical control system.

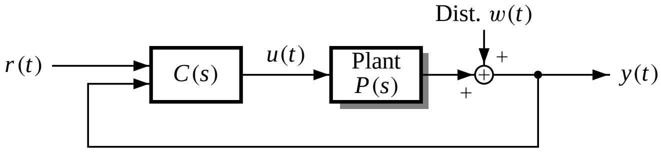


Figure 1.1: Classical control system.

The signal $r(t)$ is the reference signal. We would like the plant output $y(t)$ to track it as closely as possible. If $r(t)$ is constant or piecewise constant, the control problem is a regulator problem. If $r(t)$ varies more rapidly with time, it is a servo control problem.

To track the reference signal, the controller uses both $r(t)$ and $y(t)$ to compute the plant control signal $u(t)$. Feedback of $y(t)$ is used to stabilize the plant, and to ensure that the controller is both resilient in the face of external disturbances and able to quickly reduce the output error to zero.

Since $C(s)$ is an analog system, and subject to the vagaries of all analog computers (e.g., imprecise and drifting component values), care must be taken in its design to allow for deviation from the “ideal” transfer function. For this reason, the design may be overly conservative.

With the advent of digital computers and of digital signal processing, it became possible to design a discrete-time controller using digital hardware. Benefits of using a digital controller include primarily: the ability to implement very complex control laws, high computational precision, and great flexibility in design as the physical design (hardware) can be separated from the control algorithm design (software). Such a digital controller may be generated by discretizing the analog controller $C(s)$ (a process called *emulation*), but better results are obtained by discretizing the plant and by directly designing a digital controller $C(z)$ for the discrete-time plant. The “Plant” block in Fig. 1.1 is discretized by replacing it with a digital-to-analog (D2A) converter, followed by the original plant, and followed by an analog-to-digital (A2D) converter. The modified block diagram is shown in Fig. 1.2. From this point on, the components within the dashed box of Fig. 1.2 will be represented simply by the discretized plant, P.

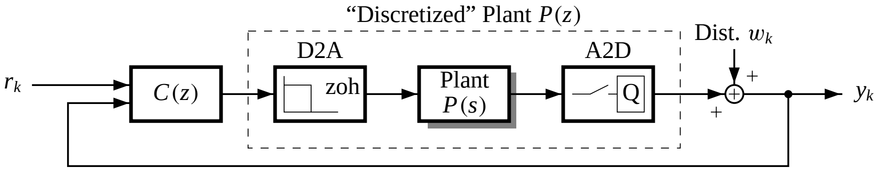


Figure 1.2: Digital control system.

Design methods for discrete-time linear controllers are well understood. Their only drawback is that they assume *precise* knowledge of the plant dynamics. For this very reason, a great deal of effort has been expended to create accurate models of "typical" plants—particularly those encountered in the military and aerospace fields. The control-system designer for industrial applications is not so fortunate. These plants are not, in general, as well understood. Consequently, post-modern control techniques concern themselves with *robust* controller design. In some sense, "optimal" performance is sacrificed in order to create a controller which will be stable and give adequate response to control inputs over a *variety* of plants $\{P\}$. These concerns are central to the context of adaptive control.

1.3 The Framework of Adaptive Inverse Control

If precise knowledge of the plant dynamics are available to the control designer, then adaptive control can be accomplished for the system in Fig. 1.2. This is shown in Chap. 4. The concept of an adaptive-controller design-process then reduces to a computer-aided-control-system-design (CACSD) tool to automatically generate a controller C to meet various design specifications. However, since CACSD tools are already available to provide optimum controllers for known linear plants, even under a variety of very complex constraints (see for example, reference [3]), adaptive control in this context is simply a novelty, and is not particularly useful.[2]

Thus, we may conclude that adaptive control is worthwhile only when the control designer *does not* have accurate information about the plant dynamics, or when those dynamics are (slowly) time-varying in an unknown way. One of the chief tasks, then, of adaptive control is that of plant

[2] A possible exception is the control of a nonlinear plant, for which much less theory exists.

identification. We will see in Chap. 5 that the framework of Fig. 1.2 is not appropriate when on-line plant identification is being performed in the presence of disturbance.

The appropriate framework for adaptive control may be developed from Fig. 1.2 in two simple steps. The first step changes the rôle of the feedback loop. Instead of feeding-back the disturbed plant output, we feed back an estimate of the disturbance. This is done following the philosophy of the *internal model control* scheme, presented in a series of papers [9, 10, 11, 34, 5, 6] by Garcia and colleagues. Figure 1.3 shows how the design paradigm changes.

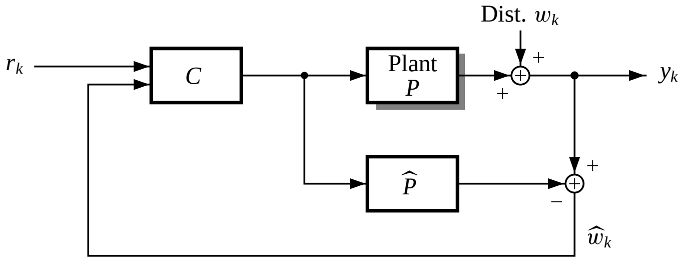


Figure 1.3: Internal model control.

A plant model $\widehat{P}$ is used to estimate the disturbance at the plant output. This estimate is fed back to the controller input. If $\widehat{P}$ is an accurate model of the plant, the control design is now effectively open-loop! A result of this is that P must be stable in order for internal model control to work. If P is originally unstable, it must be stabilized using traditional methods. The nature of the stabilization is not critical; the controller C is designed based on whatever the stabilized plant turns out to be, in order to give acceptable servo behavior. The two design objectives of stability and performance are now separated and may be handled individually rather than by joint optimization.

The second step required to make the framework suitable for adaptive control is to move the feedback path. Justification will be presented in Chap. 5. All that is required for now is to note the resulting block diagram in Fig. 1.4,[3] and especially the addition of the new filter, X. Disturbance cancelling is done by X.

The design process is now well defined. The block $\widehat{P}$ is adapted in order to model the plant; the block C is adapted to provide servo control; and the block X is adapted to perform disturbance cancelling and to ensure robustness of the design to errors in the plant model. A great deal of this dissertation will address how to properly adapt C and X.

[3] Some minor modifications to the block diagram will be required. Details will be addressed when the need arises.

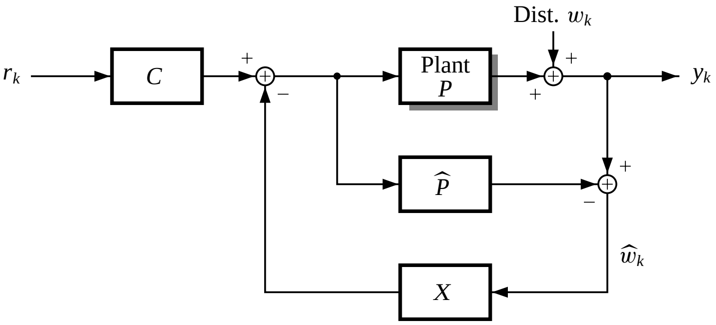


Figure 1.4: Adaptive inverse control.

1.4 Author's Contributions

The main contributions made by the author to the field of adaptive inverse control include:

1. **Constrained control of linear and nonlinear plants:** This contribution may be divided into two main areas.

   - *Analysis of constrained control*—Constraints on control effort may be required in a design due to actuator limitations. It is shown that they may also be useful to limit ringing phenomenon inherent in an inverse-control scheme. A non-adaptive method is developed to generate a controller which is guaranteed to meet constraints on the control effort. This controller may be used as an initialization for an adaptive controller design algorithm. It is also shown that the optimal constrained controller is nonlinear, even if the plant is linear. It is shown that constraints on the control effort determine a good estimate of the latency to use when controlling nonminimum phase plants.
   - *Synthesis of a constrained controller*—A gradient descent algorithm is derived to adapt a controller to minimize the mean-squared system error while meeting constraints on the control effort. The controller may be a linear FIR filter, a linear IIR filter, a nonlinear transversal filter implemented with a neural network, or a NARX filter implemented with a neural network. The algorithm is shown to decouple, so that the structure of the plant model and the structure of the controller may be chosen independently and still result in a practical design. Initialization procedures for linear controllers are presented which greatly decrease the required adaptation time. A possible initialization technique for nonlinear controllers is also proposed.

Simulation results are presented to verify the analysis and the synthesis methods. Linear and nonlinear, minimum-phase and nonminimum-phase, SISO and MIMO plants are simulated.

2. **Disturbance cancelling for linear and nonlinear plants:** This contribution may also be divided into two main areas.

    - *Analysis of disturbance cancelling*—Simultaneous on-line system identification and disturbance cancelling may come at the expense of degraded performance compared with a system which performs off-line system identification. It is shown that conventional disturbance rejection schemes fail when performed with on-line system identification. A solution, adapted from reference [45], is used to correctly perform on-line system identification and disturbance cancelling for linear plants. It is shown that the scheme still causes a degradation in performance if the plant is nonlinear, but the degradation experienced in simulation seems to be small. The structure of the function performed by the disturbance canceller is analyzed, leading to the conclusion that it is equal to a disturbance predictor cascaded with a plant inverse. The optimal predictor may be nonlinear which leads to the discovery that the disturbance canceller may need to be implemented with a nonlinear adaptive filter even if the plant is linear.
    - *Synthesis of a disturbance canceller*—Three methods are presented to generate a disturbance canceller. The first two methods require minimal or no adaptation to reach their solution, and are based on the analysis of the function performed by the disturbance canceller. The third method is based on the algorithm developed to perform constrained control, and provides the best performance, in general.

    Simulation results are presented to verify the analysis and the synthesis methods. Linear and nonlinear, minimum-phase and nonminimum-phase, SISO and MIMO plants are simulated.

3. **Compensation for sensor dynamics and sensor noise:** Previous work in adaptive inverse control has assumed that the sensors used to measure the plant output are ideal. If the sensors are not ideal, performance may be severely degraded. A method is proposed to compensate for the non-ideal sensors, and a sensitivity analysis shows that the method works well. It is found that the sensor noise and sensor dynamics affect the feedforward control and disturbance cancelling circuits differently, and the effects are analyzed. Simulation results are presented to verify the analysis.

4. **Simplification of control architecture:** In this dissertation, emphasis is put on a *systems level approach* to controller design. This translates into separable block diagrams where the structure of each block may be independently selected to optimize its task. The blocks are not coupled in any strange way.

   The prior state-of-the-art is presented in reference [45, p. 322], where five adaptive filters, twelve digital copies of the adaptive filters and six fixed digital filters are required to implement the nonlinear MIMO control scheme. The comparable method presented here (see p. 101) requires three adaptive filters, one digital copy of an adaptive filter, and one fixed digital filter. This simplification is not done at the expense of performance or higher algorithmic complexity.

1.5 Outline

This dissertation is divided into seven chapters and two appendices. Chapter 2 is a review of the adaptive filter theory required to proceed further in this work. Adaptive system identification is used as an example of how the theory is applied. Chapter 3 introduces the plants used as examples throughout the remainder of the text. Examples of linear and nonlinear, minimum-phase and nonminimum-phase, SISO and MIMO plants are included. Chapter 4 shows how to perform adaptive inverse control with constraints on the control effort. Analysis of constrained control is presented, and an algorithm to adaptively synthesize a controller is introduced. Simulations show that the scheme works very well. Chapter 5 shows how to perform disturbance cancelling with adaptive inverse control. Analysis is done to show why conventional disturbance rejection schemes do not work. More analysis demonstrates a scheme which does work, and methods are presented to adaptively synthesize a disturbance canceller. Simulations show that the methods work very well. Chapter 6 shows how to perform adaptive inverse control with imperfect sensors. Finally, chapter 7 presents conclusions and suggests future work. Appendix A introduces some analysis on stability of the LMS and backpropagation algorithms which gives a very practical method for selecting the learning rate. Appendix B presents further results of simulations for SISO nonlinear plants.

Chapter 2

Adaptive (Linear and Nonlinear) Digital Filters

There is nothing permanent except change.

—Heraclitus (c.540–c.480 B.C.)

2.1 Introduction

Adaptive inverse control is built upon the foundational theory of linear and nonlinear adaptive filtering. This chapter introduces the applicable concepts. Nothing here is new, but a few obscure attributes of these systems are reviewed as they are frequently used in later chapters.

An adaptive filter is illustrated in Fig. 2.1. It has an input, an output, and a “special input” called the desired response. The desired response d_k specifies the output we wish the filter to have. It is used to calculate an error signal e_k, which in turn is used to modify the internal parameters of the filter in such a way that the filter “learns” to perform a certain function.

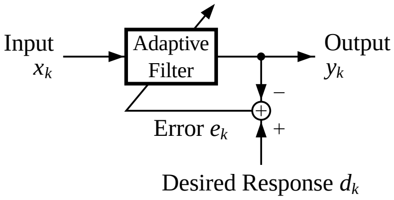


Figure 2.1: Symbolic representation of an adaptive filter.

At first, it may seem strange to have a desired response. If such a signal is available, why do we need the filter in the first place? We could just replace the filter with its desired response! For now, this question will be left unanswered. We will see later on that there are indeed various ways that such a filter may be configured such that it serves a useful purpose.

In this chapter, we will first consider linear adaptive filters—their structure, adaptation algorithm and analytic solution. Then, we look at the same characteristics for nonlinear adaptive filters. The chapter closes with an example of how to use adaptive filters to perform system identification.

2.2 Linear Adaptive Filters

2.2.1 Structure of Linear Adaptive Filters

The structure of a linear filter is illustrated in Fig. 2.2. It consists of a tapped delay line connected to the input and, possibly, a tapped delay line connected to the output. The output of the filter is calculated to be a weighted sum of the delayed inputs and outputs. The coefficients of the forward and reverse filters are termed the *weights* of the overall filter. If the weight values are fixed, the filter realizes a linear constant coefficient difference equation of the form

$$y_k - \sum_{i=1}^{N_r} w_{r,i} y_{k-i} = \sum_{i=0}^{N_f} w_{f,i} x_{k-i} . \tag{2.1}$$

Linear adaptive filters come in two basic flavors: *finite impulse response* (FIR), and *infinite impulse response* (IIR). When an FIR filter is excited by an impulse at its input, the response of the filter is non-zero for a finite period of time. An IIR filter, on the other hand, may respond with

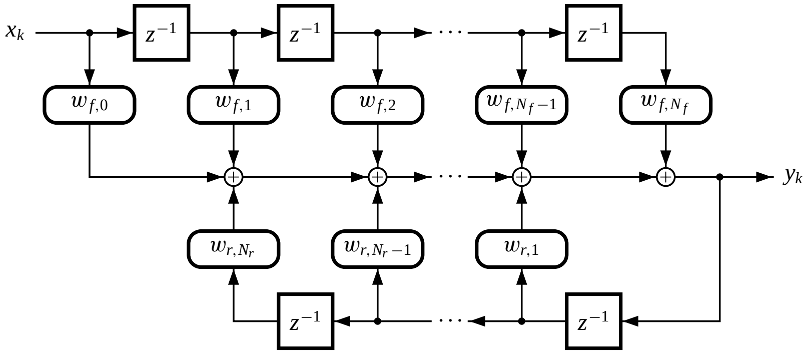


Figure 2.2: Structure of a linear filter.

non-zero values for an infinite period of time. Using the notation of Eq. (2.1), an FIR filter is one for which all the feedback weights $w_{r,i}$ are zero. An IIR filter may have non-zero $w_{r,i}$.

It is well known that any stable linear system may be approximated by a “sufficiently long” FIR filter. Therefore we concentrate on this type of filter in this dissertation. All results presented in upcoming chapters apply equally well to IIR filters, but their use was avoided due to the possibility of instability.[1] FIR filters with finite weights are always stable. IIR filters are not.

Linear MIMO filters are created via a simple extension to these ideas. A tapped delay line is connected to each input and, possibly, each output. Every output is then computed to be a weighted sum of the totality of delayed inputs and outputs.

2.2.2 Adapting Linear Adaptive Filters

Here we consider how to adapt the weights of a linear filter. At each time instant, a desired response signal d_k is applied to the filter. The true output y_k is compared to this desired response, and the error is computed to be $e_k = d_k - y_k$. After the filter has run for some time, we wish to modify the weights of the filter in order to minimize the accumulated squared error. That is, we wish to minimize the cost function J_k, where

$$J_k = \sum_{j=0}^{k} \|e_k\|^2.$$

The filter is run for k time steps, the total squared error J_k is computed, and the weights are modified by a gradient-descent optimization procedure—they are adapted in the direction of the negative gradient of the cost function with respect to the weights

$$W := W - \mu \nabla_W J_k. \tag{2.2}$$

The parameter μ is called the *learning rate*, and controls the step size in the direction of the negative gradient, and W is the vector of filter weights.

In order to calculate the gradients, we use a mathematical tool called *ordered partial derivatives*, $\partial^+(\cdot)/\partial x_k$, as opposed to the ordinary partial derivative, $\partial(\cdot)/\partial x_k$. The ordered partial derivatives were introduced by Werbos in [41], and are very well explained in [31]. They are useful for easily finding derivatives of complex dynamical systems.

A normal partial derivative of $(\cdot)$ with respect to x_k “refers to the *direct* causal impact of x_k on $(\cdot)$, while the ordered derivative refers to the *total* causal impact, including direct and indirect

[1] The use of IIR filters may be advantageous in some situations. They require fewer parameters than the equivalent FIR filter to be able to model the same system, and thus can learn with fewer data samples.

effects, both [42]." The ordered partial derivative easily calculates derivatives of equations which are evaluated *in a specific time order*. An alternate method to that of ordered partial derivatives is to substitute all intermediate equations into the final equation, and to then take ordinary total derivatives. Using ordered partial derivatives, the desired result may be found without expanding the final equation.

The primary usefulness of the ordered derivative is that complex dynamical systems may be differentiated using a simple chain rule expansion. Suppose, for example, we have a function

$$y_k = f(x_k, x_{k-1}, \ldots, x_{k-n}, W),$$

then,

$$\frac{\partial^+ y_k}{\partial W} = \frac{\partial y_k}{\partial W} + \sum_{j=0}^{n} \frac{\partial y_k}{\partial x_{k-j}} \frac{\partial^+ x_{k-j}}{\partial W}.$$

Armed with this tool, we now consider adapting an FIR (possibly MIMO) filter. The forward equation for the filter is

$$y_k = \mathbf{W} X_k,$$

where $\mathbf{W}$ is the weight matrix of the filter,[2] and X_k is a composite vector comprising all of the delayed inputs

$$X_k = [x_k^T x_{k-1}^T \ldots x_{k-N_f}^T]^T.$$

We also define the weight vector W, in terms of the weight matrix $\mathbf{W}$ to be

$$W = [\mathbf{W}_0 \mathbf{W}_1 \ldots \mathbf{W}_{N_f}]^T,$$

where, for example, $\mathbf{W}_0$ is the first row of $\mathbf{W}$.

Therefore, we can compute the gradient of the cost function with respect to the weights as follows

$$\begin{aligned} \frac{\partial^+ J_k}{\partial W} &= \sum_{j=0}^{k} \frac{\partial^+ \|e_j\|^2}{\partial W} \\ &= \sum_{j=0}^{k} -2 e_j^T \frac{\partial^+ y_j}{\partial W}, \end{aligned}$$

[2] Note that $\mathbf{W}$ and W differ. Both contain identical information, but arranged differently. The former, $\mathbf{W}$, is a *matrix*, arranged in such a way that multiplication with X_k will produce y_k. The later, W, is a *vector*, and contains the elements of $\mathbf{W}$ in an implementation-dependent arrangement. The distinction is maintained for mathematical correctness.

where,

$$\begin{aligned}\frac{\partial^+ y_k}{\partial W} &= \frac{\partial y_k}{\partial W} + \sum_{j=0}^{N_f} \frac{\partial y_k}{\partial x_{k-j}} \frac{\partial^+ x_{k-j}}{\partial W} \\ &= \operatorname{diag}\left\{X_k^T, X_k^T, \ldots X_k^T\right\} \\ &= \begin{bmatrix} X_k^T & 0 & \ldots & 0 \\ 0 & X_k^T & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & X_k^T \end{bmatrix}.\end{aligned}$$

The summation disappeared since x_j is not a function of W. Continuing,

$$\frac{\partial^+ J_k}{\partial W} = \sum_{j=0}^{k} -2e_j^T \operatorname{diag}\left\{X_j^T, X_j^T, \ldots X_j^T\right\}.$$

The way this algorithm has been presented, one would run the system for k time steps, then compute $\partial^+ J_k/\partial W$ and update the weights using Eq. (2.2). However, if the learning rate is small, and thus each weight change is small, the adaptation may be done at each time step. Then,

$$W_{k+1} = W_k + 2\mu e_k^T \operatorname{diag}\left\{X_k^T, X_k^T, \ldots X_k^T\right\}.$$

Equivalently, in our implementation-dependent, but easy-to-use format

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu e_k X_k^T.$$

This *stochastic* update rule for FIR filters is commonly known as the LMS algorithm. It is well described in several textbooks [17, 44].

2.2.3 Optimal Solution for Linear Adaptive Filters

One property of linear systems is that the mean squared error (MSE) of the system output is quadratic in the weights. This implies that there is one and only one minimum (optimal) solution when the cost function used is MSE, and that gradient descent methods will converge to the solution.[3] Another nice property is that the solution is mathematically tractable if certain statistical information about the input and desired response is available. This solution is known as the *Wiener solution.*

[3] Provided that the learning rate, μ is "small enough." See App. A for proper ranges of μ.

The details are presented in [45]. Here, we state certain properties without proof. If we let $(\phi_{xd})_n$ be the crosscorrelation function between the input x_k and the desired response d_k, and $(\phi_{xx})_n$ be the input autocorrelation function, then the unconstrained solution, $W^{(\text{opt})}(z)$, is

$$W^{(\text{opt})}(z) = \big[\Phi_{xd}(z)\big]\big[\Phi_{xx}(z)\big]^{-1},$$

where $\Phi_{xd}(z)$ and $\Phi_{xx}(z)$ are the z-transform of $(\phi_{xd})_n$ and $(\phi_{xx})_n$, respectively. Note that this solution allows for the filter $W^{(\text{opt})}$ to be non-causal. The *Shannon-Bode* solution for the optimal causal filters is

$$W^{(\text{opt})}_{\text{causal}}(z) = \big[[\Phi_{xd}(z)][\Phi^{-}_{xx}(z)]^{-1}\big]_{+}\big[\Phi^{+}_{xx}(z)\big]^{-1},$$

where, $\Phi_{xx}(z) = \Phi^{+}_{xx}(z)\Phi^{-}_{xx}(z)$ and $\Phi^{+}_{xx}(z)$ has all the poles and zeros of $\Phi_{xx}(z)$ which are inside the unit circle in the z-plane. Furthermore, the operator $[\cdot]_{+}$ means “take the time series generated by the inverse-z-transform of the operand, retain only the causal section (set the non-causal entries to zero), and take the z-transform of the result.” Mathematical analysis of a system constrained to be causal is not simple, but in certain specific cases, useful results may be obtained.

2.3 Nonlinear Adaptive Filters

2.3.1 Structure of Nonlinear Adaptive Filters

The structure of a nonlinear filter is illustrated in Fig. 2.3. It consists of a tapped delay line connected to the input and, possibly, a tapped delay line connected to the output. The output of the filter is computed to be a nonlinear function of these delayed inputs and outputs. The nonlinear function may be implemented in any way, but here we use a neural network (depicted within the dotted rectangular box).

A neural network is an interconnected set of very simple processing elements called neurons. Each neuron computes an internal sum which is equal to a constant plus the weighted sum of its inputs. The neuron outputs a nonlinear “activation” function of this sum. In this work, the activation function is chosen to be the $\tanh(\cdot)$ function

$$\text{neuron output} = \tanh\left(\text{constant} + \sum_{i=1}^{\#\text{inputs}} w_i \cdot \text{input}_i\right).$$

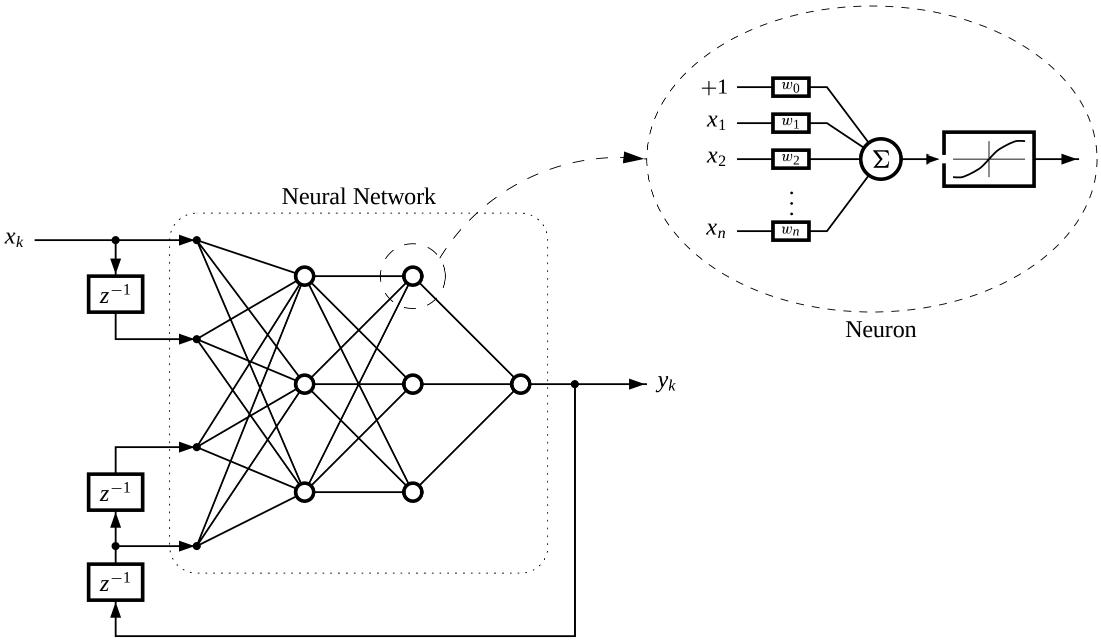


Figure 2.3: Structure of a nonlinear filter.

The constant may be easily incorporated into the summation by augmenting the input vector with a zeroth entry which is always equal to one, and by augmenting the weight vector with a zeroth entry equal to the particular desired constant. Note that in this work, all output neurons have the nonlinear function removed. This is done to give them unrestricted range.

Neural networks may be constructed from individual neurons connected in very general ways. However, it is sufficient to have *layers* of neurons, where the inputs to each neuron on a layer are identical, and equal to the collection of outputs from the previous layer (plus the augmented value "1"). The final layer of the network is called the *output layer*, and all other layers of neurons are called *hidden layers*. A layered network is a feedforward (non-recurrent) structure which computes a static nonlinear function. Dynamics are introduced via the tapped delay lines at the input to the network.

This layered structure also makes it easy to compactly describe the topology of a nonlinear filter. The following notation is used: $\mathcal{N}_{(\mathrm{a,b}):\alpha:\beta\ldots}$. This means: "The filter input is comprised of a tapped delay line with 'a' delayed copies of the exogenous input vector x_k, and 'b' delayed copies of the output vector y_k. Furthermore, there are 'α' neurons in the neural network's first layer of neurons, 'β' neurons in the second layer, and so on." For example, the filter in Fig. 2.3 would be represented as $\mathcal{N}_{(2,2):3:3:1}$. Occasionally, filters are encountered with more than one exogenous input.[4] In that case, the ‘a’ parameter is a row vector describing how many delayed copies of each input are used in the input vector to the network. For example, consider $\mathcal{N}_{([\mathrm{a}_1,\mathrm{a}_2],\mathrm{b}):\alpha:\beta\ldots}$, where ‘$\mathrm{a}_1$’ and ‘$\mathrm{a}_2$’ are the numbers of the tapped delay line copies of the exogenous inputs, and ‘b’ is the number of feedback tapped delay line copies of y_k. To avoid confusion, any strictly feedforward nonlinear filter is denoted explicitly with a zero in that part of its description. For example, $\mathcal{N}_{(2,0):3:3:1}$.

The above structure is called a *NARX* (Nonlinear AutoRegressive eXogeneous input) filter. It is general enough to approximate any nonlinear dynamical system [36]. Therefore, it is widely used in this work. Nonlinear MIMO filters are easily constructed by allowing x_k and y_k to be vectors, and by augmenting the output layer with the correct number of additional neurons.

2.3.2 Adapting Nonlinear Adaptive Filters

The weights of an adaptive nonlinear filter can be adapted using gradient descent. The methods used to adapt a feedforward network (without self-feedback) and an externally recurrent network (with feedback of the network output) differ in the details, so will be discussed separately.

Adapting a Feedforward Neural Network: A feedforward neural network may be adapted using the popular *backpropagation algorithm*, discovered independently by several researchers [41, 29], and popularized by Rumelhart, Hinton and Williams [35]. It is a method for recursively calculating the weight update for all weights in the network (for output neurons as well as hidden neurons), based on an error at the output of the network. The notation which will be used throughout this development is: $s_i^{(l)}$ is the internal sum value of neuron i in layer l, and $a_i^{(l)}$ is the output (activation value) of neuron i in layer l. For notational convenience, we denote the inputs to the network as the output of a fictitious zeroth layer of neurons: $s_i^{(0)} = a_i^{(0)} \triangleq x_i$. Furthermore, the output of the network is equal to the output of the Lth layer of neurons: $y_i \triangleq a_i^{(L)} = s_i^{(L)}$.

The network computes a function of its inputs and its weight vector

$$y_k = f(x_k, x_{k-1}, \ldots, x_{k-n}, W).$$

The difference between the actual output and the desired output is the error, and the sum of squared errors is a typical cost function to be minimized by adapting the weight vector. As we did with a

[4] An example is the disturbance cancelling filter X in Chap. 5. It may have exogeneous inputs u_k and $\widehat{w}_k$, as well as a feedback input of its own output, $\tilde{u}_k$.

linear filter, we adapt the weights in the direction of the negative gradient of the cost function with respect to the weights.

The derivation proceeds by selecting an arbitrary weight in the network, $w_{i:j}^{(l)}$—the one which connects neuron i in layer $l-1$ with neuron j in layer l—and computing the derivative of the error squared with respect to that weight.[5] Once all such derivatives are calculated, Eq. (2.2) is used to update the weights. Proceeding with the chain rule

$$\begin{aligned}\frac{\partial^{+}\|e\|^2}{\partial w_{i:j}^{(l)}} &= \frac{\partial\|e\|^2}{\partial s_j^{(l)}}\frac{\partial^{+} s_j^{(l)}}{\partial w_{i:j}^{(l)}} \\ &= \delta_j^{(l)} a_i^{(l-1)},\end{aligned}$$

where we define

$$\delta_j^{(l)} \triangleq \frac{\partial\|e\|^2}{\partial s_j^{(l)}}.$$

The values of $a_i^{(l)}$ are known from the forward pass through the network. All that remains is to calculate the $\delta_j^{(l)}$.

Since the output layer of the neural network has no activation function, $y_i = a_i^{(L)} = s_i^{(L)}$, and $\delta_i^{(L)} = -2e_i$. In the hidden layers, no specific error signal exists. We must use the chain rule expansion to determine an equivalent sensitivity of the output with respect to that weight

$$\begin{aligned}\delta_i^{(l)} &\triangleq \frac{\partial\|e\|^2}{\partial s_i^{(l)}} \\ &= \sum_j \frac{\partial\|e\|^2}{\partial s_j^{(l+1)}}\frac{\partial s_j^{(l+1)}}{\partial a_i^{(l)}}\frac{\partial a_i^{(l)}}{\partial s_i^{(l)}} \\ &= f'\big(s_i^{(l)}\big)\sum_j \delta_j^{(l+1)} w_{i:j}^{(l+1)}, \qquad l = 1, 2, \ldots L-1.\end{aligned}$$

Thus, $\delta_i^{(l)}$ is calculated by propagating values of $\delta_i^{(l+1)}$ backwards through the network. If the $\tanh(\cdot)$ activation function is used, then $f'\big(s_i^{(l)}\big) = 1 - \big(a_i^{(l)}\big)^2$, which is a particularly efficient form to compute.

The backpropagation algorithm has now been derived. In operation, a network experiences two phases: a forward phase and a reverse phase. In the forward phase, the input is propagated to

[5] A similar notation is often used in the neural network literature, but with the colon replaced by a comma. However, if one considers all of the weights in a layer to form a matrix which multiplies the inputs to that layer, the correct notation (by mathematical convention) is: $w_{j,i}^{(l)}$. That is, the i and j are reversed in the neural network literature. To help clarify this unfortunate discrepancy, a colon is used in this dissertation to distinguish the two notations.

the output to compute y_k. In the backward phase, the error is applied at the output, changed into the appropriate $\delta_i^{(L)}$ form, and the $\delta_i^{(L)}$ are propagated backward through the network. The weights are then adapted using $\delta_i^{(l)}$ and $a_i^{(l)}$

$$\Delta w_{i:j}^{(l)} = -\mu \delta_j^{(l)} a_i^{(l-1)}.$$

Calculating Jacobians of Neural Networks: Situations arise, as will be seen in the next section, where it is necessary to calculate the Jacobian of the function implemented by a neural network. Since the network is a function of its inputs and its weights, two Jacobians may be calculated. Let the composite input vector to the network be X; then, $y = f(X, W)$, and the two Jacobians are

$$\frac{\partial y}{\partial W}, \qquad \text{and} \qquad \frac{\partial y}{\partial X}.$$

They may be calculated very efficiently. The derivation is almost exactly the same as that used to determine the backpropagation algorithm. The only change is to redefine δ_i

$$\delta_i^{(l)} \triangleq v^T \frac{\partial y}{\partial s_i^{(l)}},$$

where v is an arbitrary vector. For example, instead of setting $\delta^{(L)} = -2e$, we set $\delta^{(L)} = v$. Then, we use the backpropagation algorithm to propagate these redefined δs backward through the network. If we proceed one more step, and propagate the δs *to the inputs*, and define

$$\delta_i^{(0)} \triangleq v^T \frac{\partial y}{\partial s_i^{(0)}},$$

then we have just computed

$$\left[\delta_1^{(0)}, \ldots \delta_{N_x}^{(0)}\right] = v^T \frac{\partial y_k}{\partial X}. \tag{2.3}$$

Furthermore, since $\frac{\partial s_i^{(l)}}{\partial w_{i:j}^{(l)}} = a_i^{(l-1)}$, then we have also computed, in the same step,

$$\left[\ldots, \delta_j^{(l)} a_i^{(l-1)}, \ldots\right] = v^T \frac{\partial y_k}{\partial W}, \tag{2.4}$$

where the terms in $v^T \partial y_k / \partial W$ are ordered according to the same implementation-dependent ordering of $W = [\ldots, w_{i:j}^{(l)}, \ldots]^T$.

Combining these two results, we can propagate N_y unit vectors $v = \hat{e}_i$ backward through the network to build up both Jacobian matrices, one row at a time. Werbos called this ability of the backpropagation algorithm the “dual-subroutine.” The primary subroutine is the feedforward aspect of the network. The dual subroutine is the recursive calculation of the Jacobians with the network.

Adapting an Externally-Recurrent Neural Network: Now that we have seen how to adapt a feedforward neural network and how to compute Jacobians of a neural network, it is a simple matter to extend the backpropagation algorithm to adapt externally recurrent neural networks. This was first done by Williams and Zipser [46] and called "real time recurrent learning" (RTRL). A similar presentation follows.

An externally recurrent neural network computes a function of the following form

$$y_k = f(x_k, x_{k-1}, \ldots, x_{k-n}, y_{k-1}, y_{k-2}, \ldots, y_{k-m}, W).$$

To adapt using the familiar "sum of squared error" cost function, we need to be able to calculate

$$\begin{aligned} \frac{\partial^+ \|e_k\|^2}{\partial W} &= -2e_k^T \frac{\partial^+ y_k}{\partial W} \\ \frac{\partial^+ y_k}{\partial W} &= \frac{\partial y_k}{\partial W} + \sum_{i=0}^{n} \frac{\partial y_k}{\partial x_{k-i}} \frac{\partial^+ x_{k-i}}{\partial W} + \sum_{i=1}^{m} \frac{\partial y_k}{\partial y_{k-i}} \frac{\partial^+ y_{k-i}}{\partial W}. \end{aligned}$$

The first term, $\partial y_k / \partial W$, is the direct effect of a change in the weights on y_k, and is one of the Jacobians calculated by the dual-subroutine of the backpropagation algorithm. The second term is zero, since $\partial^+ x_k / \partial W$ is zero for all k. The final term may be broken up into two parts. The first, $\partial y_k / \partial y_{k-i}$, is a component of the matrix $\partial y_k / \partial X$, as delayed versions of y_k are part of the network's input vector X. The dual-subroutine algorithm may be used to compute this. The second part, $\partial^+ y_{k-i} / \partial W$, is simply a previously calculated and stored value of $\partial^+ y_k / \partial W$. When the system is "turned on," $\partial^+ y_i / \partial W$ are set to zero for $i = 0, -1, -2, \ldots$, and the rest of the terms are calculated recursively from that point on.

Note that the dual-subroutine procedure naturally calculates the Jacobians in such a way that the weight update is done with simple matrix multiplication. Let

$$(d_w y)_k \triangleq \left[\left(\frac{\partial^+ y_{k-1}}{\partial W} \right)^T \left(\frac{\partial^+ y_{k-2}}{\partial W} \right)^T \cdots \left(\frac{\partial^+ y_{k-m}}{\partial W} \right)^T \right]^T,$$

and

$$(d_x y)_k \triangleq \left[\left(\frac{\partial y_k}{\partial y_{k-1}} \right) \left(\frac{\partial y_k}{\partial y_{k-2}} \right) \cdots \left(\frac{\partial y_k}{\partial y_{k-m}} \right) \right].$$

The latter is simply the columns of $\partial y_k / \partial X$ corresponding to the feedback inputs to the network, and is directly calculated by the dual-subroutine. Then, the weight update is calculated efficiently as

$$\Delta W = \left(2\mu e_k^T \left[\frac{\partial y_k}{\partial W} + (d_x y)_k (d_w y)_k \right] \right)^T.$$

One final comment: The reader is invited to notice that, since (in this work) the output layer of neurons is linear, a single-layer neural network is the same as a linear adaptive filter (if the bias weights are zero). The adaptation rule for a feedforward neural network, in this case, reduces to the familiar LMS rule used to adapt an FIR linear filter. The adaptation rule for an externally recurrent neural network may be used to adapt an IIR linear filter.

2.3.3 Optimal Solution for Nonlinear Adaptive Filters

In *principle*, a neural network can emulate a very general nonlinear function. It has been shown that any "smooth" static nonlinear function may be approximated by a two-layer neural network with a "sufficient" number of neurons in its hidden layer [21]. Furthermore, a NARX filter can compute any dynamical finite-state-machine (It can emulate any computer with finite memory) [36].

In *practice*, a neural network seldom achieves its full potential. Gradient-descent based training algorithms converge to a local minimum in the solution space, and not to the global minimum. However, it is instructive to exactly determine the optimal performance that could be expected from any nonlinear system, and then to use it as a lower bound on the MSE of a trained neural network. Generally, a neural network will get quite close to this bound.

The following theorem of the optimal solution is from reference [13, Theorem 4.2.1]. If the input vector to the adaptive filter is X_k, the output is y_k, and the desired response is d_k, then the optimal filter performs the function

$$y_k = \mathbb{E}\big[d_k | X_k\big].$$

This is proven by supposing that y_k is the claimed optimal estimate, and that $\hat{y}_k$ is some other estimate. We will show that $\hat{y}_k$ must yield an MSE no smaller than does y_k. To see this, consider,

$$\begin{aligned}
\text{MSE}(\hat{y}_k) &= \mathbb{E}\left[\|d_k - \hat{y}_k\|^2\right] \\
&= \mathbb{E}\left[\|d_k - y_k + y_k - \hat{y}_k\|^2\right] \\
&= \mathbb{E}\left[\|d_k - y_k\|^2\right] + \mathbb{E}\left[\|y_k - \hat{y}_k\|^2\right] + 2\mathbb{E}\left[(d_k - y_k)^T(y_k - \hat{y}_k)\right] \\
&\geq \text{MSE}(y_k) + 2\mathbb{E}\left[(d_k - y_k)^T(y_k - \hat{y}_k)\right].
\end{aligned}$$

We will show that the rightmost term is zero and hence that $\text{MSE}(\hat{y}_k) \geq \text{MSE}(y_k)$, proving the theorem. Recall that $y_k = \mathbb{E}\big[d_k | X_k\big]$ and hence

$$\mathbb{E}\big[(d_k - y_k) | X_k\big] = 0.$$

Since $y_k - \hat{y}_k$ is a deterministic function of X_k,

$$\mathbb{E}\left[(d_k - y_k)^T(y_k - \hat{y}_k) | X_k\right] = 0.$$

Then, by iterated expectation, we have

$$\begin{aligned}\mathbb{E}\big\{\mathbb{E}\big[(d_k - y_k)^T(y_k - \hat{y}_k) \,\big|\, X_k\big]\big\} &= \mathbb{E}\big[(d_k - y_k)^T(y_k - \hat{y}_k)\big] \\ &= 0,\end{aligned}$$

as claimed, which proves the theorem.

2.4 Example: Linear and Nonlinear System Identification

As an example of adaptive filtering, we now look at how to determine a plant model. The model should capture the dynamics of the plant well enough that a controller designed to control the plant model will also control the plant very well.

Such a model might be derived from physics by carefully analyzing the system and determining a set of partial-differential equations which explain its dynamics. Alternately, the model might be a "black box" implementing some sort of universal transfer function. This function may be tuned by the adjustment of its internal parameters to capture the dynamics of the system. Often, even a physically modeled system will have some adjustable parameters, so that both methods will require tuning based on observed input-output data. In the literature, this process is called *system identification*. Any method of system identification must address two major concerns:

1. We cannot have all possible "data cases" in the observed data. If we did, then system identification would be reduced to a table lookup exercise. Therefore, we must have some sort of interpolation and/or extrapolation rule.

2. We need to make some sort of assumptions about the disturbances $w(t)$ experienced by the system we are identifying. The disturbance is an *unmodeled input* to the system. If we make no assumptions about its behavior, then there is no information in the output of the plant concerning its dynamical relationship with the input.

Identification of Linear Systems: Let us first consider the identification of a linear system. We address the first concern—the interpolation/extrapolation rule—by the structure of the model. For a physically modeled system, it is readily apparent what parameters need to be identified, and how the model is structured. If we are identifying a "black box" model, its structure may be chosen to be either: a state-space implementation adjusted by subspace methods [40]; an auto-regressive ARMA IIR filter adjusted by recursive least-squares methods [24]; or an FIR model adjusted by an algorithm such as LMS [44]. Each of these methods has its advantages. The state-space method can be made numerically very robust. The ARMA model may have many fewer parameters than FIR, and hence may learn more quickly. The FIR model is very simple and unbiased by zero-mean disturbances if they are uncorrelated with the system input. The intent of this dissertation is not to address the issue of model structure selection—the reader may choose any model structure he or she may prefer. However, for the sake of brevity, adaptive FIR models are used to identify all linear systems in this text. Even if such a model is not used, the FIR impulse response of any plant model may easily be calculated by filtering an impulse and truncating the response.

We address the second concern—the character of disturbances—by assumption. For this work we assume that plant disturbances are stationary, zero-mean and uncorrelated with the plant input. This allows us to find models of the plant which are unbiased by the disturbance.

Black-box adaptive system identification is performed as shown in Fig. 2.4. The plant is excited with the signal u_k, and the disturbed output y_k is measured. The plant model $\widehat{P}$ is also excited with u_k, and its output $\hat{y}_k$ is computed. The plant modeling error is the difference between the model output and the measured plant output: $e_k^{(\text{mod})} = y_k - \hat{y}_k$. This modeling error is then used by the adaptation algorithm to update the weight values of the adaptive filter.

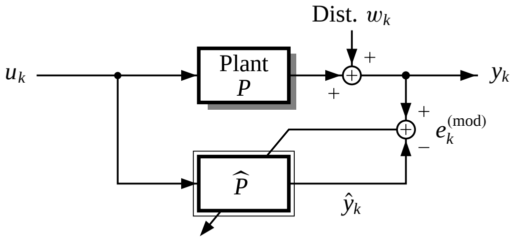


Figure 2.4: System identification.

The desired response of the filter is set to be y_k. The input to the filter is u_k. This information can be used with the techniques of Sec. 2.2.3 to compute the Wiener solution for the optimal plant model.

$$\begin{aligned}(\phi_{uy})_n &= \mathbb{E}\big[u_k y_{k+n}\big]\\ &= \mathbb{E}\big[u_k(p_{k+n} * u_{k+n} + w_{k+n})\big]\\ &= \mathbb{E}\big[u_k(p_{k+n} * u_{k+n})\big] + \mathbb{E}\big[u_k w_{k+n}\big]\\ &= p_n * (\phi_{uu})_n + (\phi_{uw})_n,\end{aligned}$$

where p_k is the impulse response of the plant. If the disturbance is zero-mean and uncorrelated with the plant input, then

$$\begin{aligned}
(\phi_{uy})_n &= p_n * (\phi_{uu})_n \\
\Phi_{uy}(z) &= P(z)\Phi_{uu}(z) \\
\widehat{P}^{(\text{opt})}(z) &= \frac{\Phi_{uy}(z)}{\Phi_{uu}(z)} \\
\widehat{P}^{(\text{opt})}(z) &= P(z).
\end{aligned}$$

So, the adaptive plant model converges to the plant.

Identification of Nonlinear Systems: Nonlinear systems are much more complicated to model. Physical modeling is still possible, but in practice this method is not as useful as “black box” modeling. For nonlinear systems, a NARX model of sufficient order is a universal dynamic system approximater. Hence, we restrict ourselves in this work to NARX neural network plant models. Neural networks have some excellent features which turn out to be very important in finding controllers; for example, the Jacobian matrices of the plant may be very simply computed. We will see that this is necessary when adapting the controller in Chap. 4.

NARX models have implicit feedback of delayed versions of their output to the input of the model (see Fig. 2.3). This feedback is assumed in all block diagrams, and is not drawn explicitly, except in Fig. 2.5. The purpose of Fig. 2.5 is to show that this feedback, when training an adaptive plant model, may be connected to either the model output $\hat{y}_k$ or the plant output y_k. The first method is called a *parallel* connection for system identification, and the second method is called a *series-parallel* method for system identification. Networks configured in the series-parallel mode may be trained using the standard backpropagation algorithm. Networks configured in the parallel mode must be trained with either real-time-recurrent-learning (RTRL) or backpropagation-through-time (BPTT). The first configuration is simple, but is biased by disturbance. The second configuration is more complex to train, but is unbiased by disturbance. In this work, nonlinear system identification is first performed using the series-parallel configuration to initialize weight values of the plant model. When the weight values converge, the plant model is re-configured in the parallel configuration and training is allowed to continue. This procedure allows speedy training of the network, but is not compromised by disturbance.

Again, we address the second concern—the character of disturbances—by assumption. We assume that plant disturbances are stationary, zero-mean and statistically independent of the plant

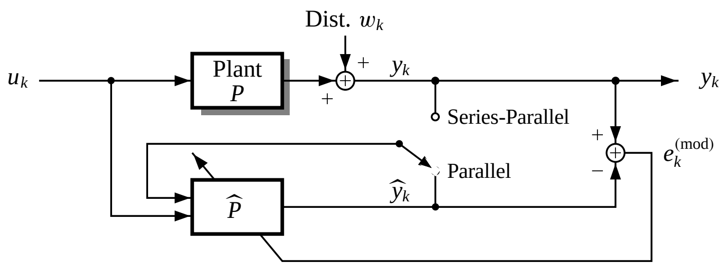


Figure 2.5: Nonlinear system identification.

input. This allows us to find models of the plant which are unbiased by the disturbance. The mathematics showing that the nonlinear system identification scheme converges to the correct result are given in Sec. 5.2.1.

2.5 Summary

This chapter has discussed linear and nonlinear adaptive digital filters. The structure, adaptation algorithm and optimal solution are given for both classes of filter. Further results concerning the stability of the adaptive algorithms may be found in App. A. An example was presented showing how to use an adaptive filter to perform adaptive system identification. The structure of the identification system was given, with proof of convergence to the correct solution provided that any disturbance is stationary zero-mean and independent of the plant input signal.

Chapter 3

Plants Used as Examples

Few things are harder to put up with than the annoyance of a good example.
—Mark Twain

3.1 Introduction

This chapter introduces the plants used throughout this dissertation as examples of their respective control categories. Representative linear and nonlinear, SISO and MIMO systems are included. The examples were chosen because they are typical of actual control problems, but simple enough to be thoroughly understood. In the following pages, the dynamics of each plant are outlined, reference signals that the plants' outputs are required to track are specified, and the characteristics of expected disturbances are presented.

3.2 Linear Plants

The dynamics of linear continuous-time systems may be expressed mathematically in a number of ways. One of these is by linear constant coefficient *differential* equations, and another is by a state-space form. In this work, the first example is defined by a differential equation, and the second by a state space form.

Continuous-time plants are discretized by realizing that the plant input is held constant for T seconds (where T is the sampling period), and the plant output is sampled every T seconds. The transfer function of the plant in the z-transform domain may then be readily calculated from the transfer function in the (Laplace) s-plane. Notationally, we say

$$H(z) = (1 - z^{-1})\ \mathcal{Z}\left\{\frac{H(s)}{s}\right\},$$

where the operator $\mathcal{Z}\{\cdot\}$ means “take the inverse Laplace transform of $(\cdot)$, sample the resulting time sequence at $1/T$ samples per second, and return the z-transform of the sampled sequence.”

The resulting transfer function $H(z)$, along with its region of convergence in the z-plane, uniquely defines a linear time invariant discrete-time system. Important properties of the system may be quickly deduced from $H(z)$. Assuming that $H(z)$ is in rational polynomial form, the roots of its denominator polynomial are called *poles*, and the roots of its numerator polynomial are called *zeros*. If all of the poles are within the unit circle in the z-plane, the system is stable and causal. If any pole is outside the unit circle, the system must be either unstable or non-causal. If all of the zeros are inside the unit circle, the system is called *minimum phase*, and a stable, causal inverse of the system exists. This makes controlling the system relatively easy. If any zero is outside the unit circle, the system is called *nonminimum phase*, and a stable, causal inverse does not exist. However, a delayed, causal, approximate inverse does exist, and works very well for controlling such systems.

Two linear plants were chosen to demonstrate aspects of adaptive inverse control in this dissertation. They are described in the following sections.

3.2.1 Linear SISO Plant

The linear SISO example was selected from reference [7, pp. 659–61]. The goal is to control the temperature of a tank of water. The flow-rate of water into the tank is constant and equal to the flow-rate of water out of the tank. The temperature of the incoming water is controlled by a mixing valve that adjusts the relative amounts of hot and cold supplies of the water (see Fig. 3.1). A length of pipe, assumed to have negligible heat loss, separates the mixing valve from the tank. This distance causes a time delay between the application of a change in the mixing valve and the discharge of the flow with the changed temperature into the tank. If our goal were to design an analog controller for this plant, this time delay significantly complicates the task. No exact analysis techniques are available to handle pure delays, and approximations, such as the Padé approximation must be used to design the controller. Discrete-time design for this plant can also be complicated. Depending on the length of the delay, the transfer function of the plant may have a finite zero outside the unit circle. Such a zero makes the plant nonminimum-phase and must be considered carefully. We will see this as we proceed.

Assuming that the mixing in the tank is instantaneous, and that negligible heat is lost in the pipe connecting the valve to the tank, the differential equation governing the tank temperature is

$$\dot{T}_t(t) + \frac{\dot{m}}{M}T_t(t) = \frac{\dot{m}}{M}T_v(t-\tau_d),$$

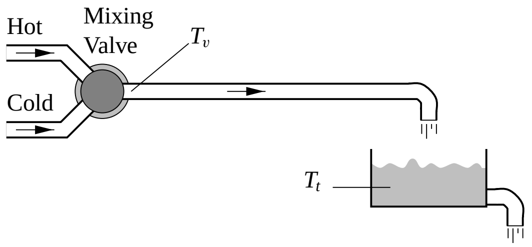


Figure 3.1: Tank temperature control.

where

$$\begin{aligned}
T_v &= \text{temperature of water immediately after the control} \\
&\quad\ \text{valve and directly controllable by the valve,} \\
T_t &= \text{tank temperature,} \\
\dot{m} &= \text{mass flow rate } (\dot{m}_{\text{in}} = \dot{m}_{\text{out}}), \\
M &= \text{water mass contained in the tank,} \\
\tau_d &= \text{delay time of water between valve and tank.}
\end{aligned}$$

When transformed, letting $a = \dot{m}/M$, the transfer function from T_v to T_t becomes

$$H(s) = \frac{e^{-\tau_d s}}{(s/a) + 1}.$$

The equivalent z-transform of the discretized plant may be computed as follows

$$H(z) = (1 - z^{-1})\ \mathcal{Z}\left\{\frac{H(s)}{s}\right\}.$$

To compute this result, we break up the delay time τ_d into integer plus fractional multiples of the sampling time T. That is, $\tau_d = \theta T - \psi T, \quad 0 \le \psi < 1$. Then,

$$\begin{aligned}
H(z) &= (1 - z^{-1})\ \mathcal{Z}\left\{\frac{e^{-\theta s T} e^{\psi s T}}{s[(s/a) + 1]}\right\} \\
&= \frac{(1 - e^{-a\psi T})}{z^{\theta}} \frac{z + \frac{e^{-a\psi T} - e^{-aT}}{1 - e^{-a\psi T}}}{z - e^{-aT}}.
\end{aligned}$$

It is interesting to consider the location of the zeros of $H(z)$. There are θ zeros located at infinity; alternately, we may consider there to be θ poles at the origin. These zeros (or poles) correspond to the built-in time delay of the system. Because of them, a non-delayed inverse cannot be constructed. However, a perfect stable and causal inverse with delay of at least θ time steps *may be* realizable.

The remaining (finite) zero of $H(z)$ will be either inside or outside the unit circle, depending on the value of ψ. If the zero is inside the unit circle, the system is generalized minimum phase. A perfect stable and causal delayed inverse may be constructed. If the zero is outside the unit circle, the system is nonminimum phase. Even if the zero is outside the unit circle, a very accurate delayed stable and causal *approximate* inverse may still be obtained. Simulations were performed for both cases to verify the efficacy of the proposed control algorithms. For these simulations, the following values were chosen for the variables

$$\begin{aligned} \dot{m} &= 2 \text{ kg/s}, \\ M &= 10 \text{ kg}, \\ T &= 1 \text{ s}, \\ \theta &= 2 \text{ time steps}. \end{aligned}$$

The value for ψ was chosen to be either 0.55 (finite zero inside unit circle) or 0.35 (finite zero outside unit circle) time steps. With all the variables substituted, the transfer functions become:

$$\begin{aligned} H_1(z) &= 0.1042\frac{z+0.7402}{z^3-0.8187z^2} \\ H_2(z) &= 0.0676\frac{z+1.6813}{z^3-0.8187z^2}, \end{aligned}$$

where $H_1(z)$ has its finite zero inside the unit circle, and $H_2(z)$ does not. These transfer functions are realized through the following difference equations, respectively:

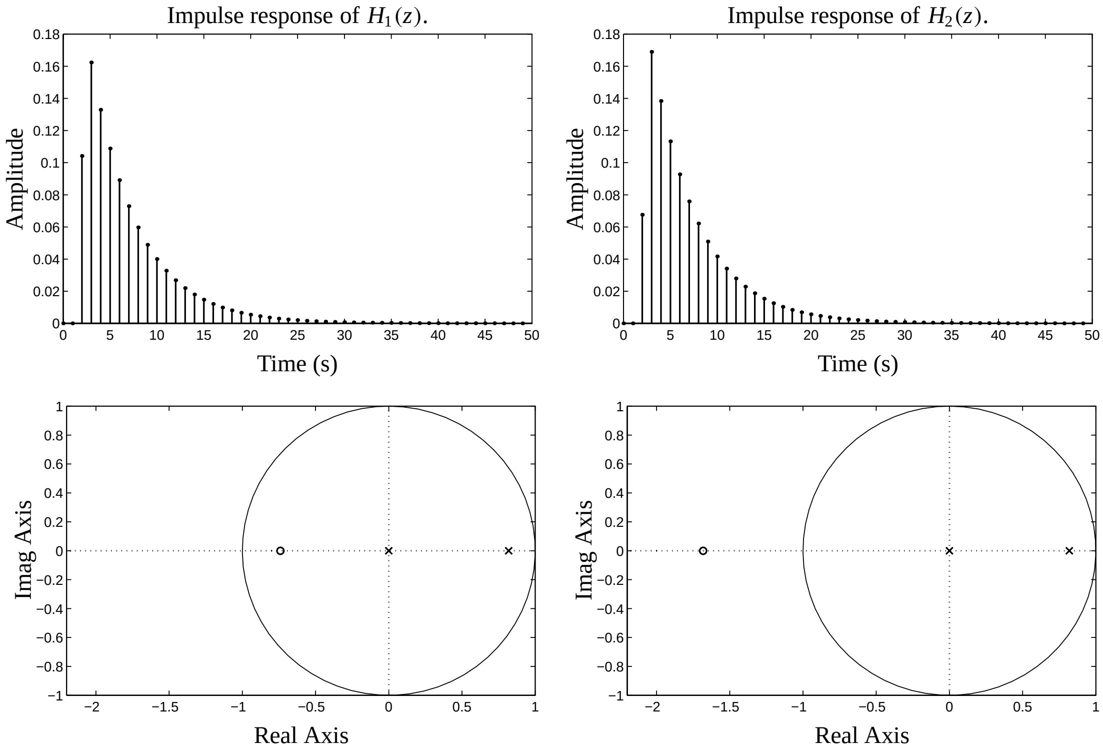
$$y_k = 0.8187y_{k-1} + 0.1042u_{k-2} + 0.0771u_{k-3}, \tag{3.1}$$

and,

$$y_k = 0.8187y_{k-1} + 0.0676u_{k-2} + 0.1137u_{k-3}. \tag{3.2}$$

For reference, the impulse responses and pole-zero plots of these transfer functions are presented in Fig. 3.2. $H_1(z)$ is referred to as the minimum-phase plant, and $H_2(z)$ is referred to as the nonminimum-phase plant.

Range of Operation: Water exists in a liquid phase (at atmospheric pressure) for temperatures between $0^\circ C$ and $100^\circ C$. To remain within the middle of this range, we would like to operate the mixer at temperatures roughly between $40^\circ C$ and $60^\circ C$. Therefore, the reference signal the plant will be required to track when performing simulations is generated by filtering i.i.d. uniformly


Impulse response of $H_1(z)$.

Impulse response of $H_2(z)$.

Figure 3.2: Discrete-time impulse responses and pole-zero plots for the linear SISO plants.

distributed random numbers (between 45°C and 55°C) using a one-pole digital filter, where the pole is located at $z = 0.7$. This is a first-order Markov process.

When constraints on the control effort are considered, they will be as follows: The control effort is allowed to be in the range 5°C to 95°C. This ensures a practical implementation as water is still in its liquid phase over this range of temperatures. Physically, this means that the hot resevoir is a 95°C hot water source, and that the cold resevoir is a 5°C cold water source.

Disturbances: There are several possible sources of disturbance for this particular plant. There could be heat loss in the pipe between the valve and the tank, heat loss in the tank itself, non-instantaneous mixing in the tank, or poorly regulated hot and cold reservoirs. It is quite reasonable to assume that the pipes and tank are very well insulated, and so heat loss is not considered to be significant. Furthermore, the relatively slow 1 Hz sampling rate allows for good mixing between samples. Therefore, we focus on the regulation of the hot and cold sources, which provides a very interesting problem.

The controller output selects a desired valve temperature. The nominal hot reservoir temperature is $T_h = 95°C$, and the nominal cold reservoir temperature is $T_c = 5°C$. Therefore, the valve is set based on the control signal, u_k, to be:[1]

$$T_v = \left(\frac{u_k - 5}{90}\right) T_h + \left(\frac{95 - u_k}{90}\right) T_c, \qquad 5 \le u_k \le 95.$$

Now, let us assume that the hot source is poorly regulated. It heats up and cools down in a periodic fashion. Let

$$T_h = 95 + 5\sin\left(2\pi t/60 + \phi\right),$$

where ϕ is random variable, uniformly distributed between $[-\pi, \pi]$, and independent of u_k.

Then,

$$\begin{aligned} T_v &= u_k + \left(\frac{u_k - 5}{18}\right) \sin\left(2\pi k/60 + \phi\right) \\ \text{dist}_k &= \left(\frac{u_k - 5}{18}\right) \sin\left(2\pi k/60 + \phi\right) * p_k, \end{aligned}$$

where the disturbance is measured at the output of the plant, so is shown convolved with the plant impulse response, p_k. This disturbance is interesting for two main reasons: (1) It is nonlinear, and (2) It is statistically dependent on u_k. Note, however, that it is *uncorrelated* with u_k.

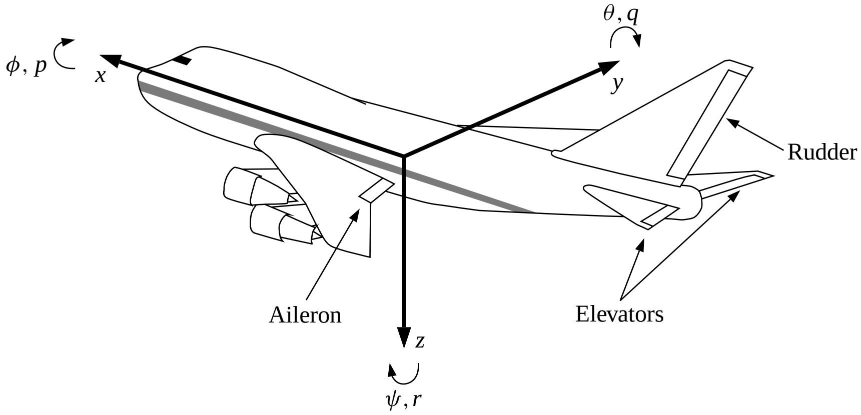
3.2.2 Linear MIMO Plant

The Boeing 747 aircraft is one of the most capable transport jets ever built (see Fig. 3.3). It can carry approximately 420 passengers and has a range of more than 8,000 miles. Because of its extensive range (resulting in pilot fatigue), and a desire to minimize the crew requirements, a capable “autopilot” controller is required in the aircraft design. With this motivation, two aspects of flight control were selected to demonstrate linear, MIMO control.[2]

As the reader might imagine, the control equations for an airplane are actually quite nonlinear; however, they may be adequately approximated by a linear model around an equilibrium point. In the case at hand, the equilibrium “point” is: level flight at 40,000 ft and a nominal forward speed of Mach 0.8 (774 ft/sec). The resulting linearized equations of motion are eighth-order, but they may be separated into two fourth-order sets representing the perturbations in longitudinal and lateral motion.

[1] Even should the control signal go outside of bounds, this relationship is used to preserve the linearity of the problem.

[2] The primary reference for this section is [7, pp. 684–93]. The author in turn references the seminal but elusive source [18]. The augmented equations for MIMO control were obtained from [14, pp. 23–35].



x, y, z	=	position coordinates
p, q, r	=	roll, pitch and yaw *rates*
ϕ, θ, ψ	=	roll (bank), pitch and yaw *angles*

Figure 3.3: Aircraft yaw-rate and bank-angle control.

The longitudinal motion consists of axial (x), vertical (z) and pitching (θ, q) motion, while the lateral motion consists of rolling (ϕ, p), yawing (ψ, r) and lateral (y) movement. Additionally, we define the side-slip angle β to be the angle between the forward velocity vector and the nose-direction of the airplane. The elevator control surfaces and the throttle control the longitudinal motion, and the aileron and rudder primarily affect lateral motion. The coupling between lateral and longitudinal motion is minimal and is usually ignored when designing controllers. Here, we wish to control the aircraft's yaw-rate (r) and bank-angle (ϕ).

The dynamics of the system are most compactly represented in state-space form. Define

$$x(t) = \left[\begin{array}{l} \text{Sideslip angle, } \beta(t)\text{, in radians} \\ \text{Yaw rate, } r(t)\text{, in radians/second} \\ \text{Roll rate, } p(t)\text{, in radians/second} \\ \text{Bank angle, } \phi(t)\text{, in radians} \end{array}\right]$$

$$y(t) = \left[\begin{array}{l} \text{Yaw rate, } r(t)\text{, in radians/second} \\ \text{Bank angle, } \phi(t)\text{, in radians} \end{array}\right],$$

and,

$$u(t) = \left[\begin{array}{l} \text{Rudder angle in degrees} \\ \text{Aileron angle in degrees} \end{array}\right].$$

Then,

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t), \end{aligned}$$

where,

$$A = \begin{bmatrix} -0.0558 & -0.9968 & 0.0802 & 0.0415 \\ 0.5980 & -0.1150 & -0.0318 & 0.0000 \\ -3.0500 & 0.3880 & -0.4650 & 0.0000 \\ 0.0000 & 0.0805 & 1.0000 & 0.0000 \end{bmatrix}, \qquad B = \begin{bmatrix} 0.0073 & 0.0000 \\ -0.4750 & 0.1230 \\ 0.1530 & 1.0630 \\ 0.0000 & 0.0000 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The set of continuous-time impulse responses corresponding to this set of equations is plotted in Fig. 3.4. As can be seen, the decay rate of these impulse responses is very slow, with some effect even after 500 seconds. In order to improve this we use output feedback to increase the damping on the very lightly-damped modes of the system. The very simple feedback structure is shown in Fig. 3.5. As will be shown shortly, the response time is greatly improved, and a simpler digital controller (*i.e.*, one with fewer taps) may be realized.

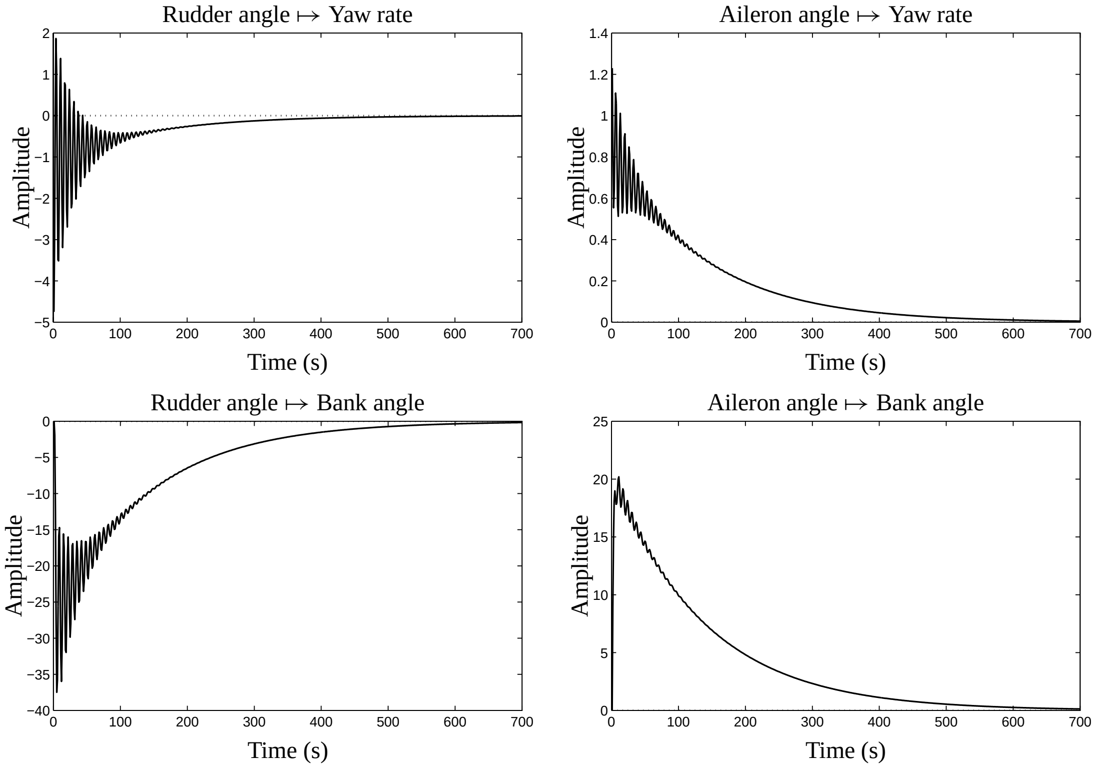


Figure 3.4: Uncompensated jet impulse responses.

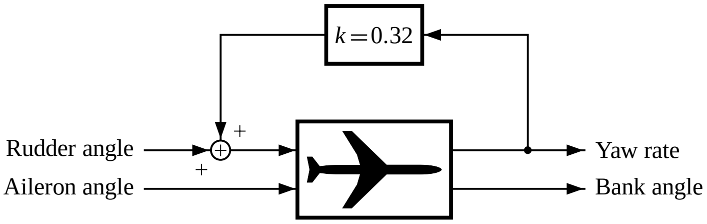


Figure 3.5: Simple feedback system to increase the damping of one of the lateral modes.

The process of converting this system from continuous-time to discrete-time is simple yet tedious. Fortunately, the *Control Systems Toolbox for Matlab* [14] comes to our rescue and gives us the following set of equations (for $T = 0.5$ seconds)

$$x_k = \begin{bmatrix} \text{Sideslip angle, } \beta_k \text{, in radians} \\ \text{Yaw rate, } r_k \text{, in radians/second} \\ \text{Roll rate, } p_k \text{, in radians/second} \\ \text{Bank angle, } \phi_k \text{, in radians} \end{bmatrix},$$

$$y_k = \begin{bmatrix} \text{Yaw rate, } r_k \text{, in radians/second} \\ \text{Bank angle, } \phi_k \text{, in radians} \end{bmatrix},$$

and,

$$u_k = \begin{bmatrix} \text{Rudder angle in degrees} \\ \text{Aileron angle in degrees} \end{bmatrix}.$$

Then,

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_k \\ y_k &= C_d x_k, \end{aligned}$$

where,

$$A_d = \begin{bmatrix} 0.8876 & -0.3081 & 0.0415 & 0.0198 \\ 0.2020 & 0.3973 & -0.0046 & 0.0024 \\ -1.2515 & 0.5106 & 0.7617 & -0.0139 \\ -0.3313 & 0.1510 & 0.4407 & 0.9976 \end{bmatrix}, \quad B_d = \begin{bmatrix} 0.4806 & -0.0013 \\ -1.5809 & 0.3887 \\ 0.0599 & 4.8390 \\ 0.0390 & 1.2585 \end{bmatrix},$$

$$C_d = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The corresponding discrete-time impulse responses are shown in Fig. 3.6. Note in particular that the output feedback has reduced the length of the impulse responses to about 25 seconds.

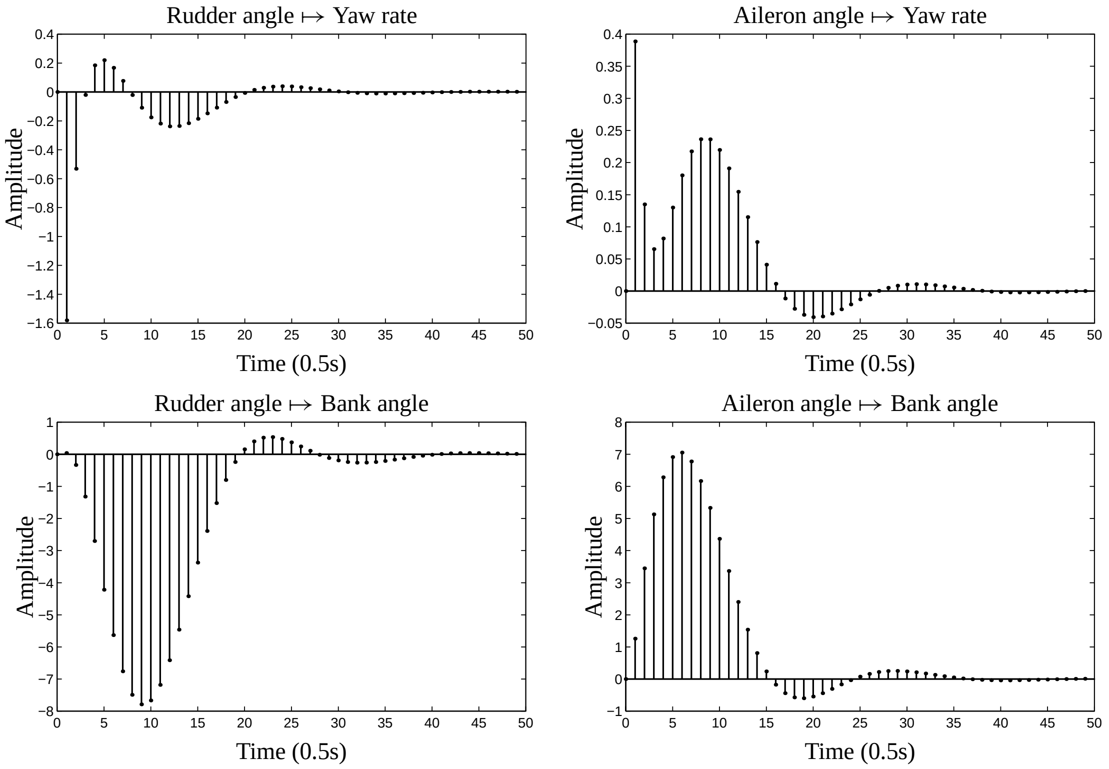


Figure 3.6: Compensated, discrete-time jet impulse responses.

Range of Operation: Proper ranges for the input and output signals of this plant are unknown, so the following limits were arbitrarily chosen. The reference signal for the yaw-rate to track varied between ±0.1 radians per second. The bank angle to be tracked varied between ±0.4 radians. This corresponds roughly to yaw-rates between ±6° per second, and bank angles between ±23°. The reference command to be tracked was generated independently for each output. The reference command for the desired yaw-rate was a first-order Markov process generated by filtering i.i.d. uniform random variables with maximum value 0.03 using a one-pole filter whose pole was at $z = 0.9$. The reference command for the desired bank angle was a first-order Markov process generated by filtering i.i.d. uniform random variables with maximum value 0.12 with a one-pole filter whose pole was $z = 0.9$.

Constraints on the control effort were considered to be slew-rate constraints. The slew-rate of the rudder angle was constrained to be between ±0.3° per second. The slew-rate of the aileron angle was constrained to be between ±1.5° per second.

Disturbances: The primary disturbance experienced by the dynamics of the airplane are those induced by bursts of wind. It is assumed here that the nominal wind values are incorporated into the dynamic model of flight, and that gusts around that nominal value are the disturbances. The state of the airplane, x_k, is affected directly by the wind. So, the full discrete-time model of the airplane dynamics, with disturbance, is

$$\begin{aligned} x_{k+1} &= A_d\left[x_k + \mathrm{dist}_k\right] + B_d u_k \\ y_k &= C_d x_k, \end{aligned} \tag{3.3}$$

Furthermore, it is assumed that the wind gusts occur as planar fronts and thus do not affect the yaw-rate, roll-rate or bank-angle directly. Instead, the sideslip angle is directly affected by the wind, and the other state-variables are affected indirectly through the dynamical relationship between themselves and the sideslip angle. If we model the wind in the lateral direction, then the sideslip angle is perturbed by

$$\tan^{-1}\left(\frac{\text{wind speed}}{\text{airplane speed}}\right).$$

The model for generating a wind speed time series was derived based on the data presented in [20]. An approximation was made to the autocorrelation function of the cited paper. The power spectral density of wind velocity was calculated from the autocorrelation function, and was found to be

$$\Phi(f) = \frac{3950}{1 + (20\pi f)^2}.$$

An FIR filter was designed using a weighted least-squares optimization algorithm to produce this power spectral density given an input stream of i.i.d. uniform random numbers with maximum magnitude 1. The filter impulse response and the power spectral density of wind disturbances is shown in Fig. 3.7. The maximum absolute wind speed is in the neighborhood of 20 feet per second, so the maximum perturbation to β_k is around 0.03 radians.

3.3 Nonlinear Plants

Unlike linear systems, nonlinear systems do not satisfy the superposition principle. Therefore, they cannot be described in terms of impulse responses or transfer functions. They must be described in the time domain. Continuous time systems may be described by systems of nonlinear *differential* equations, and discrete time systems may be described with sets of nonlinear *difference* equations.

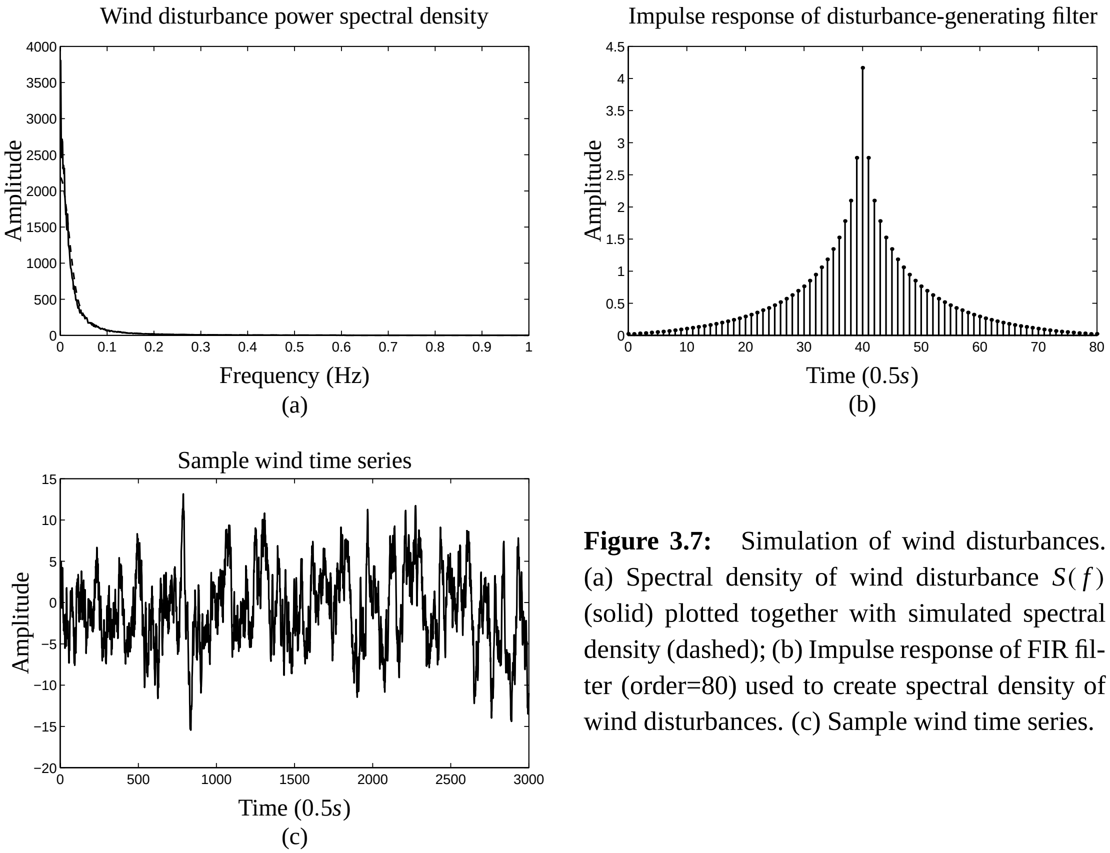


Figure 3.7: Simulation of wind disturbances. (a) Spectral density of wind disturbance $S(f)$ (solid) plotted together with simulated spectral density (dashed); (b) Impulse response of FIR filter (order=80) used to create spectral density of wind disturbances. (c) Sample wind time series.

It is not always possible to analytically discretize a set of nonlinear differential equations. In many cases it is necessary to discretize the plant by simulating (numerically integrating) the differential equations over the sampling period. At times, dozens of integration steps need to be taken to advance the system from its current state to its state after a sampling period.

Some work has already been done in the area of nonlinear adaptive inverse control [2, 4]. This work focused on seven nonlinear SISO plants, all defined by difference equations. Successful feedforward control was achieved for most, but not all, of the plants. In this work, these seven plants are considered in App. B to demonstrate that the methods presented in later chapters are able to control all seven plants, and that disturbance cancelling may also be achieved for them.

Here, however, we consider two more difficult nonlinear control problems with greater practical motivation. Much more insight may be gathered from studying them since we already have an expectation of what their limits of performance might be. Their dynamics are computed by simulating the continuous-time differential equations since they cannot be analytically discretized.

3.3.1 Nonlinear SISO Plant

Autopilots for ships are often designed to keep the ship's heading (yaw angle) in a desired direction (see Fig. 3.8). There are some applications, such as course changing and turning, however, where it is desirable to be able to track a time-varying reference direction. This scenario was selected as an example of a very nonlinear control problem. The primary reference is [39] but [30] was also consulted for additional insight into the meanings of some of the parameters involved.

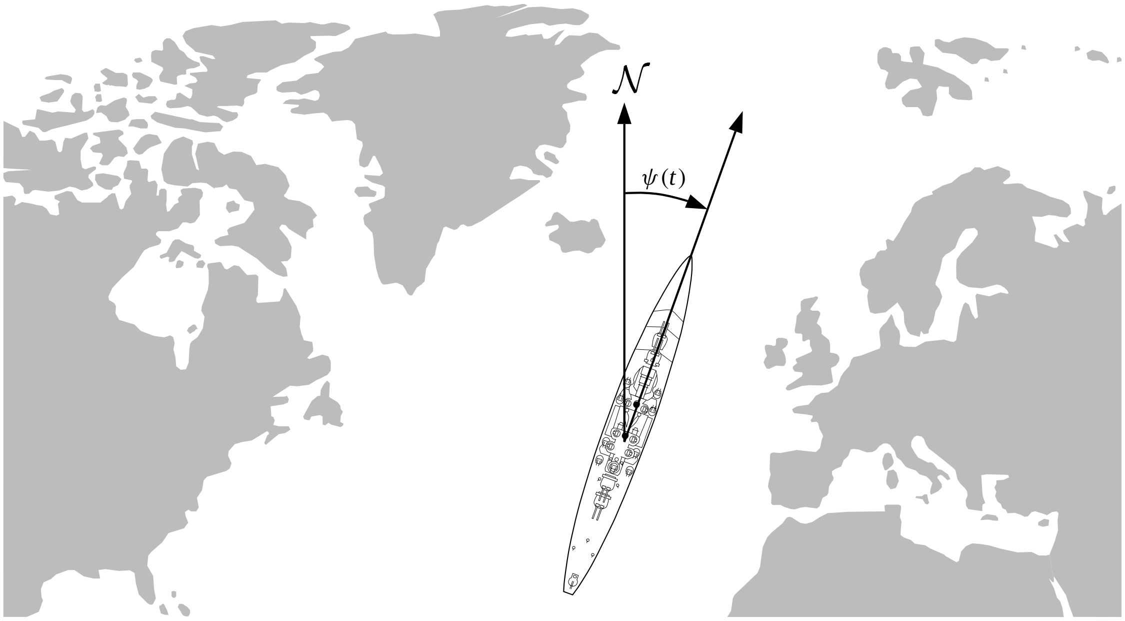


Figure 3.8: Illustration of heading (yaw) angle, $\psi(t)$. Not drawn to scale!

The SISO maneuvering model of a ship may be expressed as

$$\ddot{\psi} + k\, d(\dot{\psi}) = k\delta_r,$$

where $\psi(t)$ is the yaw angle of the ship, δ_r is the rudder angle and $d(\dot{\psi})$ is a damping term of the form

$$d(\dot{\psi}) = d_3\dot{\psi}^3 + d_2\dot{\psi}^2 + d_1\dot{\psi} + d_0.$$

Because of symmetry, most ships have the property that $d_2 = d_0 = 0$.

Not only does the ship itself exhibit a nonlinear dynamical relationship between its heading and rudder angle, but so too does the rudder angle with respect to the (steering) wheel position; that is, the rudder angle δ_r does not follow the wheel angle δ_w exactly. The rudder is rate-limited to 6° per second until $|\delta_w - \delta_r| \le 3^\circ$ after which the rudder operates in the linear range of its characteristic. One final restriction is that the rudder angle may not exceed 35° in either direction. Keeping all these things in mind, the ship dynamics may be represented as shown in Fig. 3.9.

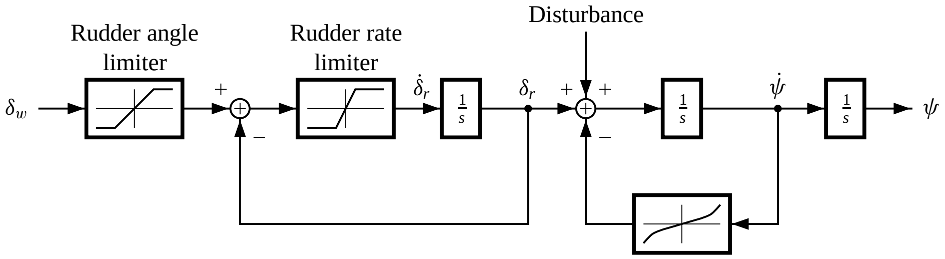


Figure 3.9: Block diagram of ship yaw dynamics from wheel angle δ_w to heading angle ψ.

To summarize, the system dynamics are controlled by the coupled pair of differential equations

$$\begin{aligned}\ddot{\psi}(t) &= k\left[\delta_r(t) + \text{disturbance}(t) - d(\dot{\psi}(t))\right]\\ \dot{\delta}_r(t) &= 6\text{sat}\left[\frac{35\text{sat}\left(\frac{\delta_w(t)}{35}\right) - \delta_r(t)}{3}\right],\end{aligned}$$

where,

$$\text{sat}(x) = \begin{cases} -1, & x < -1;\\ x, & -1 \le x < 1;\\ 1, & \text{otherwise,}\end{cases}$$

and constants have been added to convert from degrees to radians and to allow the use of the normalized saturation function $\text{sat}(x)$. The sampling rate used in the discrete time controller was 2Hz. All remaining parameters were taken from reference [39].

$$k = 0.0107, \qquad d_1 = 9.42, \qquad d_3 = 2.24,$$

and correspond to the dynamics of a Royal Navy warship traveling at sixteen knots.

Stabilizing the Dynamics: The dynamics of the ship are unstable. This may easily be seen by applying a step function to the control input. The plant output for a collection of step inputs ranging in magnitude from 10° to 180° is shown in Fig. 3.10(a). A bounded input does not produce a bounded output, and hence the dynamics are unstable.

A very simple feedback circuit can stabilize the dynamics. The modification to the ship block diagram is shown in Fig. 3.11. The step responses for the stabilized dynamics are shown in Fig. 3.10(b). Now, bounded inputs produce bounded outputs. In fact, the feedback loop makes a pretty good control system all by itself. A step input of ψ° will asymptotically produce a step output of ψ°. The nonlinear controller will enhance the dynamic response where it can (when the rudder rate and angle limits are not saturated).

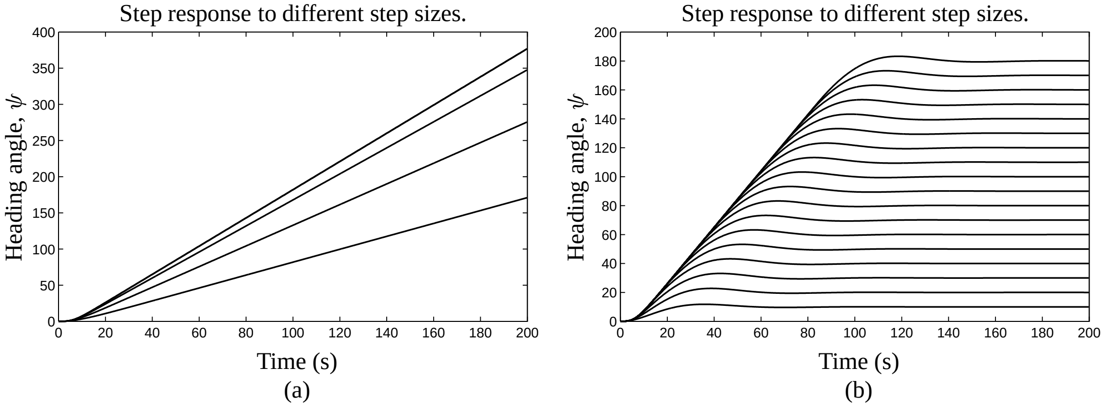


Figure 3.10: Step response of ship to input step commands of magnitude 10° to 180° in increments of 10°. (a) Unstable ship dynamics; (b) Stabilized ship dynamics.

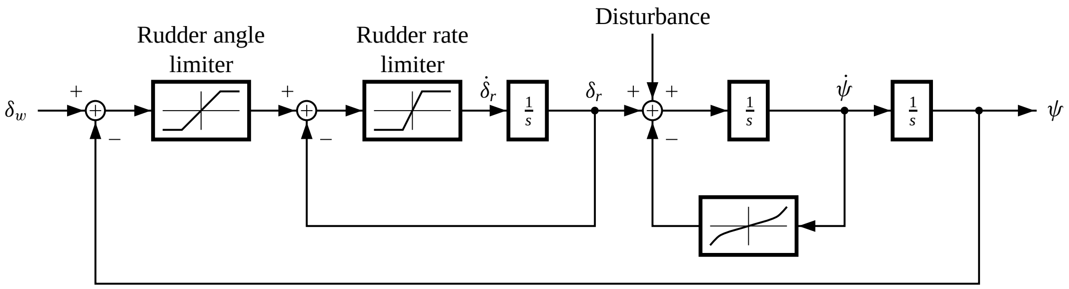


Figure 3.11: Block diagram of stabilized ship.

Range of Operation: Since the nonlinear controller can not improve performance when the rudder dynamics are in their saturation region, only relatively small perturbations around a fixed heading need to be considered. A default “steady” heading of 0° was used, with perturbations limited to ±30° around that heading. More specifically, the reference command to be tracked was a first-order Markov process, generated by filtering i.i.d. uniform random numbers with maximum magnitude 0.05° using a one-pole filter with the pole at $z = 0.99$.

Disturbances: The disturbances experienced in the dynamics of the ship are caused almost exclusively by the action of sea waves acting on the rudder angle, and by wind acting on the superstructure. Here, we consider only the effects of waves, as is done in reference [39]. The power spectral density of wave height as a function of wave frequency is

$$S(f) = \frac{\alpha g^2}{(2\pi)^4 f^5} \exp\left[-\beta \left\{\frac{g}{2\pi U_{19.5} f}\right\}^4\right],$$

where,

$$\begin{aligned} f &= \text{wave frequency (Hz)} \\ \alpha &= \text{Phillips constant } (8.1 \times 10^{-3}) \\ \beta &= \text{dimensionless constant } (0.74) \\ U_{19.5} &= \text{wind velocity 19.5m above sea level (knots)} \\ g &= \text{acceleration due to gravity}. \end{aligned}$$

The nominal wind velocity $U_{19.5}$ was taken to be 20 knots. This power spectral density (scaled as will be described later) is plotted as the solid line in Fig. 3.12(a).

In order to generate wave disturbances, i.i.d. uniformly distributed random numbers with maximum magnitude 1 are passed through a filter having the same power spectral density $S(f)$. This filter, shown in Fig. 3.12(b), was designed from $S(f)$ using a least-squares filter design method. As is done in reference [39], the filter is scaled so that peak-to-peak yaw rates of approximately 0.3° per second occurred when there was no rudder input. The scaling factor was 3. Using the random uniform input and this filter, a sample wave time series is plotted in Fig. 3.12(c).

3.3.2 Nonlinear MIMO Plant

The plant chosen to illustrate nonlinear MIMO control is a two-link planar robot arm. This is a standard example from the robotics literature; the equations of motion are taken from reference [25] and the specific parameter values are taken from reference [38]. The model includes all joint coupling terms (centripetal and Coriolis torques, variable effective moments of inertia, etc.).

The manipulator is depicted in Fig. 3.13. Both links are capable of 360° rotation. As the emphasis of this dissertation is not on robotic control, we do not present a detailed tutorial on the subject. However, a few details and definitions will be mentioned to aid the discussion.

There are two major aspects to robotic control. The first is the *kinematics* of the manipulator. These specify the relationship between the angles of the joints and the x, y location of the end

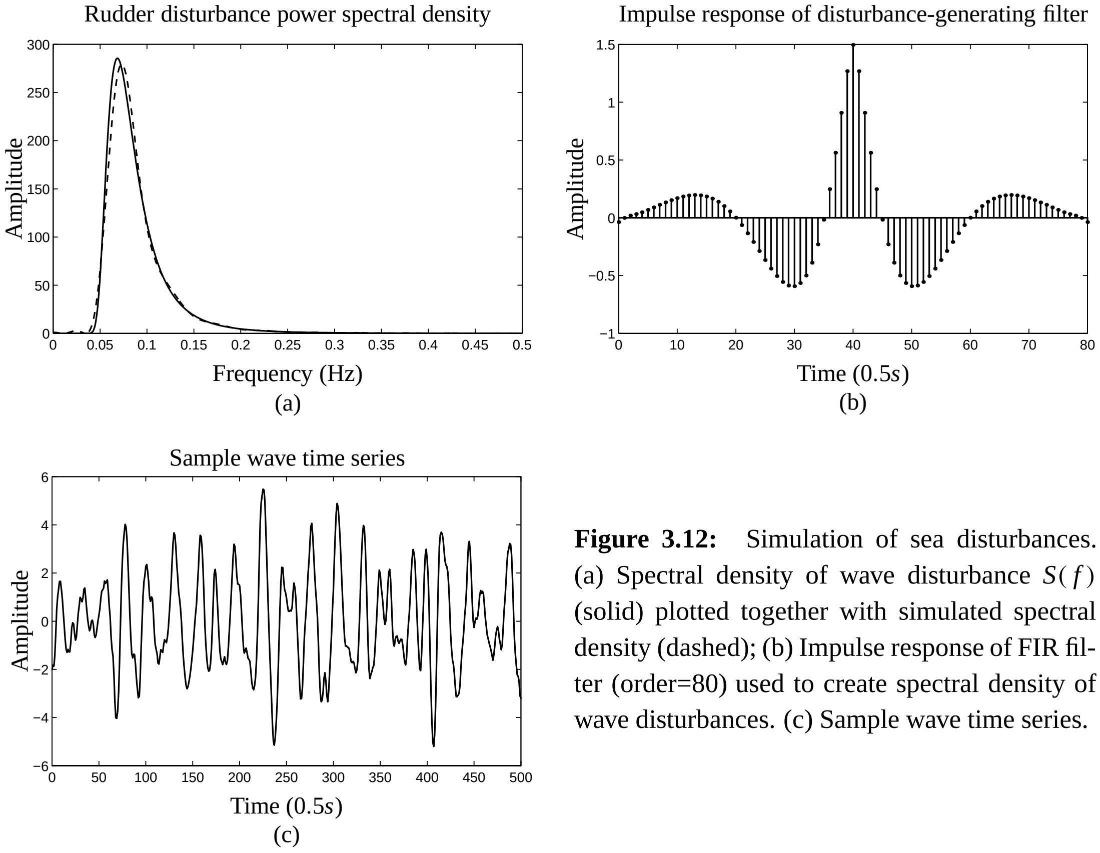


Figure 3.12: Simulation of sea disturbances. (a) Spectral density of wave disturbance $S(f)$ (solid) plotted together with simulated spectral density (dashed); (b) Impulse response of FIR filter (order=80) used to create spectral density of wave disturbances. (c) Sample wave time series.

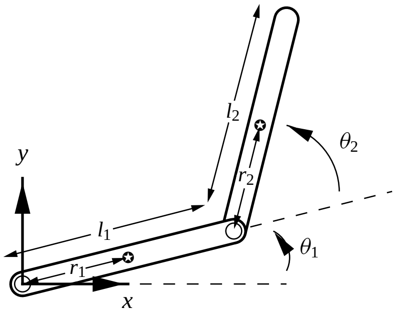


Figure 3.13: Two-link robot arm.

effector. The so-called *inverse kinematics* problem concerns itself with finding an appropriate set of joint angles to achieve some desired end-effector position. There may be no solution at all to a specific inverse kinematics problem, but in general there are multiple solutions. Neural networks have been used to solve the inverse kinematics problem in a slightly different application [27, 12].

The second aspect to robotic control concerns the *dynamics* of the manipulator. These are concerned with computing the forces required to achieve a certain movement of the robot. Inertial, Coriolis, centrifugal, and gravitational forces are considered. Some advanced models also account for frictional forces and for the flexibility of the manipulator, which is usually considered a rigid body when the physics are developed. The standard Lagrangian model of manipulator dynamics is given by

$$\tau = \mathcal{A}(\theta)\ddot{\theta} + \mathcal{B}(\theta)\left[\dot{\theta}\dot{\theta}\right] + \mathcal{C}(\theta)\left[\dot{\theta}^2\right] + \mathcal{G}(\theta),$$

where,

$\mathcal{A}(\theta)$	is the $n \times n$ kinetic energy matrix,
$\mathcal{B}(\theta)$	is the $n \times n(n-1)/2$ matrix of Coriolis torques,
$\mathcal{C}(\theta)$	is the $n \times n$ matrix of centrifugal torques,
$\mathcal{G}(\theta)$	is the n-vector of gravity torques,
$\ddot{\theta}$	is the n-vector of joint-angle accelerations,
$\dot{\theta}$	is the n-vector of joint-angle velocities,
θ	is the n-vector of joint angles,
τ	is the joint-force vector.

Also, the symbols $\left[\dot{\theta}\dot{\theta}\right]$ and $\left[\dot{\theta}^2\right]$ are notation for the $n(n-1)/2$-vector of velocity products and the n-vector of squared velocities. They are defined as

$$\begin{aligned}\left[\dot{\theta}\dot{\theta}\right] &= \left[\dot{\theta}_1\dot{\theta}_2, \dot{\theta}_1\dot{\theta}_3 \ldots \dot{\theta}_1\dot{\theta}_n, \dot{\theta}_2\dot{\theta}_3, \dot{\theta}_2\dot{\theta}_4 \ldots \dot{\theta}_{n-2}\dot{\theta}_n, \dot{\theta}_{n-1}\dot{\theta}_n\right]^T \\ \left[\dot{\theta}^2\right] &= \left[\dot{\theta}_1^2, \dot{\theta}_2^2 \ldots \dot{\theta}_n^2\right]^T.\end{aligned}$$

As can be seen, the matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{G}$ are all dependent on the configuration of the manipulator θ.

These equations have been expanded and simplified for this particular manipulator

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \alpha + 2\beta c_2 & \delta + \beta c_2 \\ \delta + \beta c_2 & \delta \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} -\beta s_2\dot{\theta}_2 & -\beta s_2(\dot{\theta}_1 + \dot{\theta}_2) \\ \beta s_2\dot{\theta}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix},$$

where $c_2 = \cos(\theta_2)$, $s_2 = \sin(\theta_2)$ and

$$\begin{aligned}\alpha &= I_{z1} + I_{z2} + m_1 r_1^2 + m_2(l_1^2 + r_2^2) \\ \beta &= m_2 l_1 r_2 \\ \delta &= I_{z2} + m_2 r_2^2,\end{aligned}$$

and: $I_{z(\cdot)}$ is the z-component of the inertial tensor of the $(\cdot)$th link around its center of mass (relative to a frame attached at the center of mass of the link and aligned with the principle axes of the bar), $m_{(\cdot)}$ are the link masses, $r_{(\cdot)}$ is the distance from the joint to the center of mass of the link, and $l_{(\cdot)}$ is the length of the link.

The values of the parameters are:

$$\begin{aligned}
m_1 &= 1 \text{ kg} \\
m_2 &= 2 \text{ kg} \\
l_1 &= 1 \text{ m} \\
l_2 &= 1.2 \text{ m} \\
r_1 &= 0.5 \text{ m} \\
r_2 &= \frac{l_2}{2} + \frac{0.5 m_3 l_2}{m_2 + m_3} \text{ m} \\
I_{z1} &= 0.12 \text{ kg}\,\text{m}^2 \\
I_{z2} &= \frac{m_2 l_2^2}{12} + m_2 \left(r_2 - \frac{l_2}{2} \right)^2 + m_3 (r_2 - l_2)^2 \text{ kg}\,\text{m}^2.
\end{aligned}$$

The third mass, m_3, is a point-mass load at the end of the second link. It is considered to have a value of 0 kg under normal circumstances. A more involved discussion follows, under the section of disturbances. The sampling rate for discrete-time control was 100 Hz.

Stabilizing the Dynamics: The dynamics of the robot are not globally stable. The dynamics were stabilized by creating a PD controller.

$$\tau_k = -K_p (y_k - u_k) - K_d \dot{y}_k,$$

where, $K_p = 2000I$ and $K_d = 100I$. This is shown in Fig. 3.14.

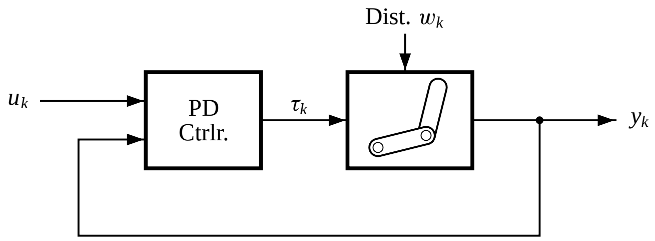


Figure 3.14: Stabilized robot.

Range of Operation: The two links of the arm are required to follow independent reference commands. Both commands are generated by filtering i.i.d. uniform random variables with maximum magnitude 0.15π using a one-pole filter whose pole was at $z = 0.95$. Slew-rate constraints on the control input required that τ_1 be between $\pm 15 N \cdot m/10ms$, and τ_2 be between $\pm 5 N \cdot m/10ms$.

Disturbances: The "disturbance" experienced by the robotic arm is due to a time-varying load m_3 attached to the end of the manipulator. The load is allowed to vary between 0 kg and 10 kg, with its value chosen in discrete increments of 0.1 kg. As a function of time, the load is piecewise constant. The duration that a specific load is attached to the end of the manipulator is a geometric random variable with average 100 samples (1 second).

3.4 Summary

This chapter introduces four specific plants to be used throughout this dissertation as representative of their specific control classes. There is a linear SISO plant, a linear MIMO plant, a nonlinear SISO plant and a nonlinear MIMO plant. Additionally, the linear SISO plant may be selected to be minimum phase or nonminimum phase by changing one of its parameters. All simulations done in this dissertation (with the exception of those done for further examples presented in App. B) are performed to demonstrate control of these plants.

In each case, the plant dynamics are discussed, the reference signals they are required to track are specified and the disturbances experienced by the plant are characterized. The reader who wishes to duplicate any result in this dissertation should have enough information at hand to do so!

Chapter 4

Constrained Adaptive Feedforward Control

Who controls the past controls the future.
Who controls the present controls the past.
—George Orwell in *1984*

4.1 Introduction

To perform adaptive inverse control, we need to be able to adapt the three filters of Fig. 1.4: the plant model $\widehat{P}$, the controller C, and the disturbance canceller X. One of these tasks has already been addressed—we saw in Sec. 2.4 how to adapt $\widehat{P}$ to make a plant model. For the time being we set aside consideration of the disturbance cancelling filter X and concentrate on the design of the feedforward controller C. This chapter presents an algorithm which can be used to train C to perform constrained model-reference based control of a linear or nonlinear, SISO or MIMO plant. This algorithm has a number of nice properties and works very well. Along the way, some new analytical results relating to constrained adaptive inverse control of linear plants are presented, and some insight into choosing the delay for the control of a nonminimum-phase plant is discovered.

From time to time, results obtained from simulation are presented to corroborate and illuminate certain analytical results before the algorithm used in the simulation is developed. The reader is asked to accept the fact that the algorithm will be developed later in the chapter.

This chapter is organized into three parts. The first part deals with analytical results pertaining to constrained control, particularly for linear MIMO (and hence also linear SISO) plants. The second part develops an algorithm to train a controller to perform constrained control, and discusses an efficient implementation. The third part presents results from simulations for the plants of Chap. 3.

4.2 Analysis of Constrained Linear Control

4.2.1 A Working Linear MIMO Control Architecture

The notion of the "optimal controller" is very important in this chapter. Various degrees of optimality are considered. Without any constraints, the optimal controller is the one which minimizes the mean-squared system error: $\mathbb{E}\big[(d_k - y_k)^T(d_k - y_k)\big]$. A controller may not achieve this level of performance due to constraints on its architecture. For example, we might restrict the controller to be a linear system, or to be causal. In that context, we still talk about the "optimal controller" as being the one which minimizes the mean-squared system error while satisfying the architecture constraints. We know from Chap. 2 that the optimal solution for a linear system is the Wiener solution, and that the optimal solution for a causal linear system is the Shannon-Bode solution.

In addition to imposing constraints on the architecture of the controller, we might impose constraints on the output of the controller; that is, on the control effort u_k. Determining the controller weights for a linear controller with control-effort constraints is a convex optimization problem and closed-form solutions for the transfer function of the controller are not available. This chapter presents a simple analytic method to determine the weights of the controller to meet the control-effort constraints—but not necessarily minimize the mean-squared system error—and an algorithm which adapts a controller to find the optimal constrained solution which *does* minimize mean-squared system error. Before exploring the adaptive algorithm, it is worthwhile to consider the analytic solution for the linear causal controller with and without constraints on the control effort.

Linear SISO and MIMO plants may be controlled using linear SISO or MIMO controllers. Here, we look at a method for adapting the weights of a linear causal controller. This controller satisfies architecture constraints but does not consider constraints on the control effort—a different algorithm is developed later on for that purpose. The controller which satisfies only architecture constraints will be called the "optimal causal controller" and denoted $C^{(\text{opt})}_{\text{causal}}(z)$.

The optimal causal linear MIMO controller is adapted using a simplification of a method from reference [45, Chap. 10]. This method is diagrammed in Fig. 4.1. An on-line feedforward part controls the plant, and two off-line processes are used to adapt the controller. The off-line process on the left makes a left-inverse $V(z)$ of the plant model $\widehat{P}(z)$, and the process on the right uses this left-inverse to make a model-reference based right-inverse $C(z)$ of $\widehat{P}(z)$. A copy of $C(z)$ is then used as the system controller. If $\widehat{P}(z)$ is equal to $P(z)$, and $V(z)$ is a good left-inverse of $\widehat{P}(z)$, then the resulting controller minimizes the mean-squared system error.

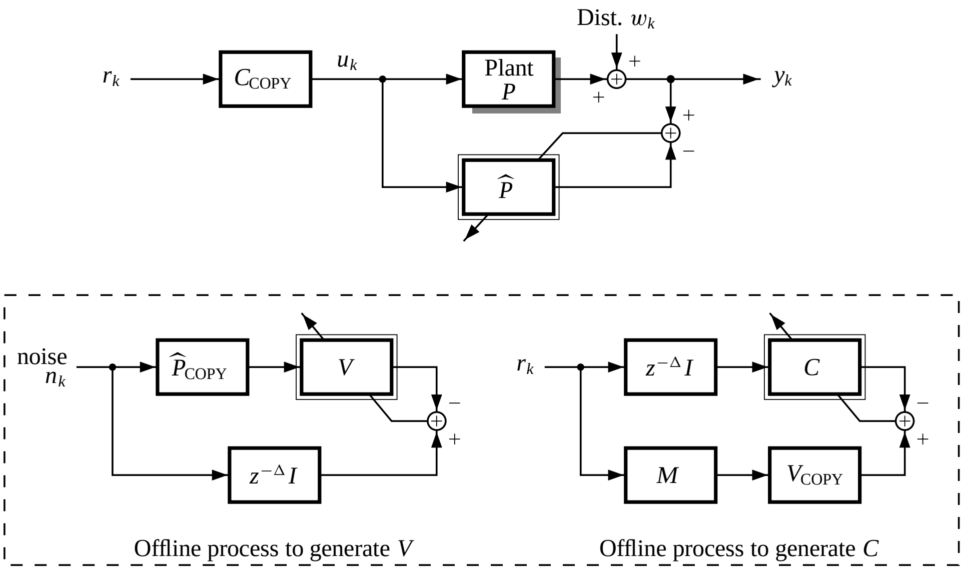


Figure 4.1: MIMO controller design.

The off-line process on the left makes $V(z)$, a left-inverse of $\widehat{P}(z)$. If the plant is minimum-phase, then the delay Δ may be set to zero. If the plant is nonminimum-phase, then the delay is set long enough to make an excellent delayed inverse of $\widehat{P}(z)$. This delay is removed when adapting the controller, so it does not affect the latency of the final control system. There is no penalty for selecting a large delay if it is necessary to create a good inverse.

Since we assume Δ to be sufficiently large, the transfer function of the adaptive filter $V(z)$ will approach its unconstrained Wiener solution. This may be calculated using the techniques of Chap. 2.

$$\begin{aligned}
\Phi_{xd}^{(V)}(z) &= \big(z^{-\Delta}I\big)\big(\widehat{P}_{\text{COPY}}(z^{-1})\big)\Phi_{nn}(z) \\
\Phi_{xx}^{(V)}(z) &= \big(\widehat{P}_{\text{COPY}}(z)\big)\big(\widehat{P}_{\text{COPY}}(z^{-1})\big)\Phi_{nn}(z) \\
V^{(\text{opt})}(z) &= \big(\Phi_{xd}(z)\big)\big(\Phi_{xx}(z)\big)^{-1} \\
&= \big(z^{-\Delta}I\big)\big(\widehat{P}_{\text{COPY}}(z)\big)^{-1}
\end{aligned}$$

The off-line process on the right of the figure uses a copy of $V(z)$ to adapt the controller, $C(z)$. The delay Δ is the same as in the off-line process used to generate $V(z)$. The reference model $M(z)$ specifies the desired transfer function of the controlled system, including any delay required to make a good inverse of a nonminimum-phase plant. The user chooses this delay to strike a compromise between system latency and precision of control. We must use the Shannon-Bode approach to compute the optimal causal solution for $C(z)$ (since the plant may be nonminimum-phase, the Wiener solution may be noncausal).

$$\begin{aligned}
\Phi_{xd}^{(C)}(z) &= \left(z^{+\Delta} I\right) V(z) M(z) \Phi_{rr}(z) \\
&= \left(\widehat{P}_{\text{COPY}}(z)\right)^{-1} M(z) \Phi_{rr}(z) \\
\Phi_{xx}^{(C)}(z) &= \Phi_{rr}(z) \\
C_{\text{causal}}^{(\text{opt})}(z) &= \left[\left(\Phi_{xd}(z)\right)\left(\Phi_{xx}^{-}(z)\right)^{-1}\right]_{+} \left(\Phi_{xx}^{+}(z)\right)^{-1} \\
&= \left[\left(\widehat{P}_{\text{COPY}}(z)\right)^{-1} M(z) \Phi_{rr}^{+}(z)\right]_{+} \left(\Phi_{rr}^{+}(z)\right)^{-1} \qquad (4.1)
\end{aligned}$$

If the plant is minimum-phase, then this simplifies to:

$$C_{\text{causal}}^{(\text{opt})}(z) = \left(\widehat{P}_{\text{COPY}}(z)\right)^{-1} M(z).$$

The analysis in the following sections computes a controller which satisfies constraints on the control effort. This analysis uses $C_{\text{causal}}^{(\text{opt})}(z)$ as a starting point.

4.2.2 Constraint on the Control Effort

The preceding section sketches a method to adapt a controller to perform model-reference based control while minimizing mean-squared system error. In general, however, we would also like to be able to incorporate actuator constraints in the design process. Reasons include, (summarizing [3, p. 190]):

- *Saturation.* Exceeding absolute limits on actuator signals may damage an actuator, or cause the plant model $\widehat{P}$ to be a poor model of the system being controlled. This may be avoided by constraining the peak magnitude of u_k.

- *Actuator heating.* Persistently large actuator signals may cause excessive heating and damage the system. This may be avoided by constraining the RMS norm of u_k.

- *Power, fuel, or resource use.* Large actuator signals may be associated with excessive resource use. This may be avoided by constraining the average-absolute norm of u_k.

- *Mechanical wear.* Excessively rapid changes in the actuator signal may cause undesired stresses or excessive wear. This may be avoided by constraining the slew rate of u_k.

One further reason we might like to constrain the control effort is more subtle. We are performing *discrete-time* control of *continuous-time* plants. If pure inverse control is performed, then the plant output will match the desired output very well *at the sampling times*. However, the plant output may behave very poorly during the inter-sample interval. Figure 4.2 shows an example. An inverse controller was trained for the minimum-phase tank example of Sec. 3.2.1, with the reference model $M(z) = z^{-2}$. The step response of the controlled system is shown in Fig. 4.2(a). The discrete-time response of the system is shown as dots, and the continuous-time response of the system is shown as a solid line. The desired response is shown as a gray line. We see that the discrete-time response precisely tracks the desired response. However, due to very high control effort, the continuous-time response "rings" wildly about the desired response. A second controller was trained with exactly the same desired response, but which incorporated constraints on the peak magnitude of the control signal u_k. The step-response of the controlled system is shown in Fig. 4.2(b). Although the rise time is longer, the settling time is shorter, and a much smoother continuous-time step response is achieved. Further details of this simulation may be found in Sec. 4.4.1.

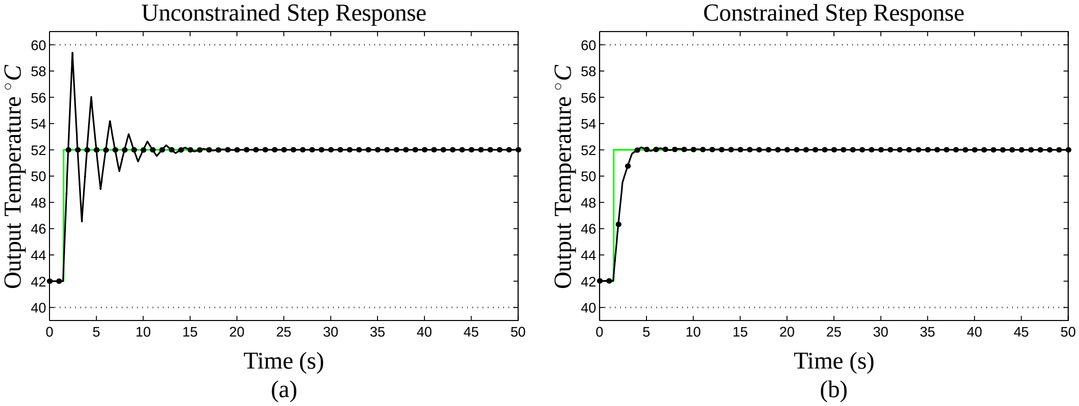


Figure 4.2: Step response of controlled minimum-phase tank. In (a), the control effort is unconstrained. In (b), the control effort is constrained to be between 5°C and 95°C. The dots are the discrete-time response; the solid line is the continuous-time response; and the gray line is the desired response. (The control effort is plotted in Fig. 4.18(b) and Fig. 4.18(f).)

4.2.3 An Approximate Solution for the Constrained Controller

Constraints on the control effort are most naturally expressed in the time-domain. On the other hand, the Wiener and Shannon-Bode methods for determining optimal solutions for controllers use the frequency-domain. As such, it is not clear how the constraints on control effort will change the solution for the controller itself.

This section presents some analysis which translates constraints on the *control signal* u_k in the time domain into constraints on the *controller* C in the frequency domain. The controller and plant are assumed to be linear—possibly MIMO—systems. The result is a "recipe" for turning an unconstrained controller into one which is *guaranteed* to satisfy the constraints. The design is conservative, but an example is presented which demonstrates that the solution is quite good. First, we examine constraints on the peak magnitude of the control signal u_k, and then extend the analysis to encompass constraints on the peak slew rate.

Constraints on the Peak Control Effort

Here, we consider designing a controller such that its peak output over all time is bounded by some user-specified constant, $u_k^{(\text{peak})}$. If the controller has more than one output, then we restrict the peak of each individual output to be below $u_k^{(\text{peak})}$. Mathematically, we express the peak output over all time using the infinity-norm of the control signal

$$\|u_k\|_\infty \triangleq \sup_k \max_i |u_{k,i}|,$$

where u_k is the vector control signal at time k, and $u_{k,i}$ is the ith component of that control signal. Then, the constraint on the control effort becomes

$$\|u_k\|_\infty \leq u_k^{(\text{peak})}.$$

We convert this time-domain specification into a frequency-domain specification using some simple results relating time-domain bounds to frequency-domain bounds. Each inequality is "tight" in the sense that equality is achievable with some set of input signals.

$$\begin{aligned}
\|u_k\|_\infty &\leq \|U(e^{j\omega})\|_\infty \\
&\triangleq \sup_{\omega\in[-\pi,\pi]} \|U(e^{j\omega})\|_2 \\
&= \sup_{\omega\in[-\pi,\pi]} \|C(e^{j\omega})R(e^{j\omega})\|_2 \\
&\leq \sup_{\omega\in[-\pi,\pi]} \bar{\sigma}[C(e^{j\omega})]\|R(e^{j\omega})\|_2.
\end{aligned}$$

The last line replaces the norm of $C(e^{j\omega})$ with its maximum singular value, $\bar{\sigma}[C(e^{j\omega})]$, found using a singular-value decomposition. Since we want $\|u_k\|_\infty \leq u_k^{(\text{peak})}$, this means

$$\bar{\sigma}[C(e^{j\omega})] \leq \gamma(e^{j\omega}) \triangleq \frac{u_k^{(\text{peak})}}{\|R(e^{j\omega})\|_2}, \qquad \forall\,\omega. \tag{4.2}$$

Any controller which satisfies this constraint will have acceptable control effort. Any controller which does not satisfy this constraint may or may not have acceptable control effort.

Equation (4.2) may be used to design a controller which is *guaranteed* to satisfy the time-domain restrictions on u_k. To do this, we modify the unconstrained controller $C^{(\mathrm{opt})}_{\mathrm{causal}}(z)$ designed in Sec. 4.2.1.

The controller $C^{(\mathrm{opt})}_{\mathrm{causal}}(z)$ may not reduce the system error to zero. Any residual error will be called the *irreducible error*. If we were to use a different controller, by definition sub-optimal, then there will be additional error at the system output. This will be called the *reducible error*. When we design a controller to meet constraints on the control effort, we need to satisfy Eq. (4.2) while minimizing the added reducible error.

If $y_k^{(\mathrm{opt})}$ is the (undisturbed) system output using the optimal controller, and y_k is the (undisturbed) system output using the sub-optimal controller, then the square-reducible error, summed over time, is

$$\begin{aligned}\sum_{k=0}^{\infty}\left(e_k^{(\mathrm{reducible})}\right)^T\left(e_k^{(\mathrm{reducible})}\right) &= \sum_{k=0}^{\infty}\left(y_k^{(\mathrm{opt})}-y_k\right)^T\left(y_k^{(\mathrm{opt})}-y_k\right)\\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\mathrm{Tr}\left[\left(Y^{(\mathrm{opt})}(e^{j\omega})-Y(e^{j\omega})\right)^*\left(Y^{(\mathrm{opt})}(e^{j\omega})-Y(e^{j\omega})\right)\right]d\omega\\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\left\|Y^{(\mathrm{opt})}(e^{j\omega})-Y(e^{j\omega})\right\|_2^2 d\omega\end{aligned}$$

where the second line is Parseval's relationship for MIMO systems, and the third line is due to the fact that the trace of a scalar is just that scalar. This function may be minimized by independently minimizing its value at each frequency. Note that

$$\begin{aligned}Y^{(\mathrm{opt})}(e^{j\omega}) &= P(e^{j\omega})C^{(\mathrm{opt})}_{\mathrm{causal}}(e^{j\omega})R(e^{j\omega})\\ Y(e^{j\omega}) &= P(e^{j\omega})C(e^{j\omega})R(e^{j\omega})\end{aligned}$$

Let

$$C^{(\mathrm{opt})}_{\mathrm{causal}}(e^{j\omega}) = \upsilon(e^{j\omega})\Sigma(e^{j\omega})\nu^*(e^{j\omega})$$

where $\upsilon(e^{j\omega})$, $\Sigma(e^{j\omega})$, and $\nu(e^{j\omega})$ form the singular-value decomposition of $C^{(\mathrm{opt})}_{\mathrm{causal}}(e^{j\omega})$. Then, also let

$$C(e^{j\omega}) = \upsilon(e^{j\omega})\Lambda(e^{j\omega})\nu^*(e^{j\omega})$$

where $\upsilon(e^{j\omega})$ and $\nu(e^{j\omega})$ are the same, and $\Lambda(e^{j\omega})$ is used to design C. The design goal is now to minimize

$$\left\|P(e^{j\omega})\upsilon(e^{j\omega})\left\{\Sigma(e^{j\omega})-\Lambda(e^{j\omega})\right\}\nu^*(e^{j\omega})R(e^{j\omega})\right\|^2$$

while satisfying the constraint of Eq. (4.2). If $\Lambda(e^{j\omega})$ is chosen to be diagonal with positive entries, then this is very easy to do. Each diagonal entry of $\Lambda(e^{j\omega})$ is a singular value of $C(e^{j\omega})$. It is very easy to see then if Eq. (4.2) is satisfied. The entries of $\Lambda(e^{j\omega})$ are chosen as follows:

$$\Lambda_i(e^{j\omega}) = \begin{cases} \Sigma_i(e^{j\omega}), & \text{if } \Sigma_i(e^{j\omega}) \leq \gamma(e^{j\omega}); \\ \gamma(e^{j\omega}), & \text{otherwise.} \end{cases} \tag{4.3}$$

We now have a "recipe" for designing a controller which will minimize the system mean-squared error while satisfying a constraint on the maximum control effort. It is summarized in Fig. 4.3.

```
begin {Generate constrained C}
        Determine $R(e^{j\omega})$ and then find $\gamma(e^{j\omega})$ using Eq. (4.2).
        Design the optimal unconstrained controller $C^{(\mathrm{opt})}_{\mathrm{causal}}(z)$ using the method in
            Sec. 4.2.1.
        At each frequency, perform the singular-value decomposition of $C^{(\mathrm{opt})}_{\mathrm{causal}}(z)$ to find
            $\upsilon(e^{j\omega})$, $\Sigma(e^{j\omega})$ and $\nu(e^{j\omega})$.
        Determine $\Lambda(e^{j\omega})$ from $\Sigma(e^{j\omega})$ and $\gamma(e^{j\omega})$ using Eq. (4.3).
        Compute $C(e^{j\omega}) = \upsilon(e^{j\omega})\Lambda(e^{j\omega})\nu^*(e^{j\omega})$.
        The impulse response for implementing $C$ may be found by taking the inverse
            FFT of $C(e^{j\omega})$.
end {Generate constrained C}
```

Figure 4.3: An algorithm for synthesizing a linear MIMO controller to satisfy constraints on the control effort.

Constraints on the Peak Slew Rate

The slew rate s_k of the system is computed as $s_k = u_k - u_{k-1}$. This is the same as passing the control signal u_k through the filter $\left(1 - z^{-1}\right)I$. This filter has a gain of two. Therefore, to construct a controller which satisfies constraints on the slew rate, we use the exact same analysis as above, except that Eq. (4.2) becomes

$$\bar{\sigma}[C(e^{j\omega})] \leq \gamma(e^{j\omega}) \triangleq \frac{s_k^{(\mathrm{peak})}}{2\|R(e^{j\omega})\|_2}, \qquad \forall\,\omega,$$

because of this extra factor of two. This new value of $\gamma(e^{j\omega})$ may be used to design a controller using the algorithm of Fig. 4.3 to satisfy the slew-rate constraints.

Example of Constraints on Peak Control Effort

At this point it is profitable to see an example of constrained control. A simulation was performed, using the algorithm to be developed later in this chapter, to find the unconstrained inverse of the minimum-phase tank of Sec. 3.2.1. A simulation was also performed to find the linear constrained inverse when the control effort was restricted to be between $5^\circ C$ and $95^\circ C$. The input source was equal to the constant $50^\circ C$ plus a first-order Markov process with pole at $z = 0.7$ and driven by i.i.d. uniform random variables with zero mean and maximum magnitude $5^\circ C$. For the purposes of the analysis here, since the system is linear, the input and output are shifted down by $50^\circ C$ to change the constraints such that $\|u_k\|_\infty \leq 45^\circ C$.

The results of these simulations are shown in Fig. 4.4. This figure plots the magnitude response of the unconstrained inverse as a solid line, the magnitude response of the constrained inverse as a dashed line, and $\gamma(e^{j\omega})$ as a gray line. The design procedure for finding a controller which is guaranteed to meet the constraints is to set the magnitude response of the controller equal to the minimum of the unconstrained magnitude response and $\gamma(e^{j\omega})$ at each frequency. The adaptive algorithm has done something very similar, but not exactly this. Since the design procedure of Fig. 4.3 is conservative, the adaptive controller is able to do somewhat better.

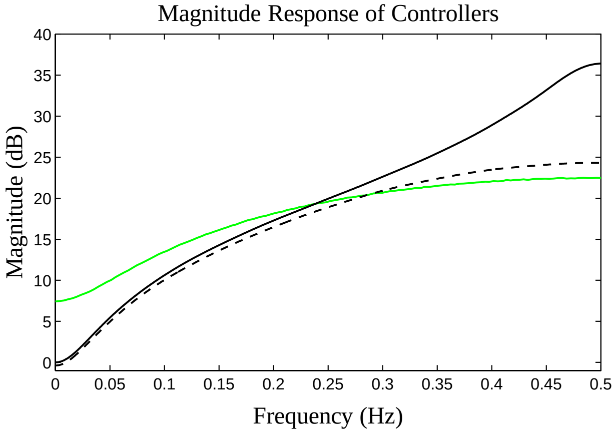


Figure 4.4: Magnitude response of unconstrained (solid line) and constrained controllers (dashed line), plotted with $\gamma(e^{j\omega})$ (gray line).

The Optimal Constrained Controller is Nonlinear

Before concluding this section, it should be mentioned that all of the analysis has been for a linear controller controlling a linear plant. This may work relatively well but, in general, optimal constrained control is performed by a nonlinear system. This may be proven by (1) assuming that the optimum controller is linear, and (2) providing a counter-example.

Suppose that the plant has trivial dynamics $P(z) = 1$, and that the reference model is $M(z) = 1$. Let the input signal, r_k, be equal to 0.1 with probability p, and equal to 10 with probability $1 - p$. Constrain the control effort such that $\|u_k\|_\infty \le 1$.

The optimal unconstrained controller is $C^{(\mathrm{opt})}_{\mathrm{causal}}(z) = 1$. The optimal linear constrained controller is $C(z) = 1/10$. The optimal nonlinear constrained controller is $u_k = \mathrm{sat}(r_k)$, where $\mathrm{sat}(\cdot)$ is the saturation function. In this case, the unconstrained controller has mean-squared error of zero. The linear controller has mean-squared error of $p \cdot (0.0081) + (1 - p) \cdot (81)$. The nonlinear controller has mean-squared error $(1-p)\cdot(81)$. The nonlinear controller is always better than the linear controller, and becomes relatively more so as $p \to 1$. A general rule of thumb is that if the input signal is very regular, then the linear controller does well. If the input signal has infrequent “spikes,” then the nonlinear controller may be much better. Figure 4.7 shows an example of the nonlinear controller giving better performance than the linear controller when controlling the nonminimum-phase tank from Sec. 3.2.1. The simulations themselves are described in a future section. Another simulation was done for the minimum-phase tank from Sec. 3.2.1. Histograms of the control effort were computed when using a linear controller and when using a nonlinear controller. These histograms are shown in Fig. 4.5. It is easily seen that the nonlinear controller makes better use of the available control effort. The nonlinear controller also had significantly better mean-squared system error.

4.2.4 Control Effort and Controlling Nonminimum-Phase Plants

If the plant to be controlled is nonminimum-phase, then a stable and causal inverse does not exist. A delayed inverse must be used as the controller. The longer the delay, the lower the system error (as will be shown). The designer is then left with the question of how to pick the “best” delay. The application itself may determine this factor by specifying a certain maximum latency; however, we can not assume that this is always the case.

When there is a constraint on the control effort, the designer has some help choosing the system latency. There exists a value of delay beyond which the control ceases to provide better performance. This value of delay can be thought of as the “best” delay if there are no other guidelines. We now proceed to show this mathematically.

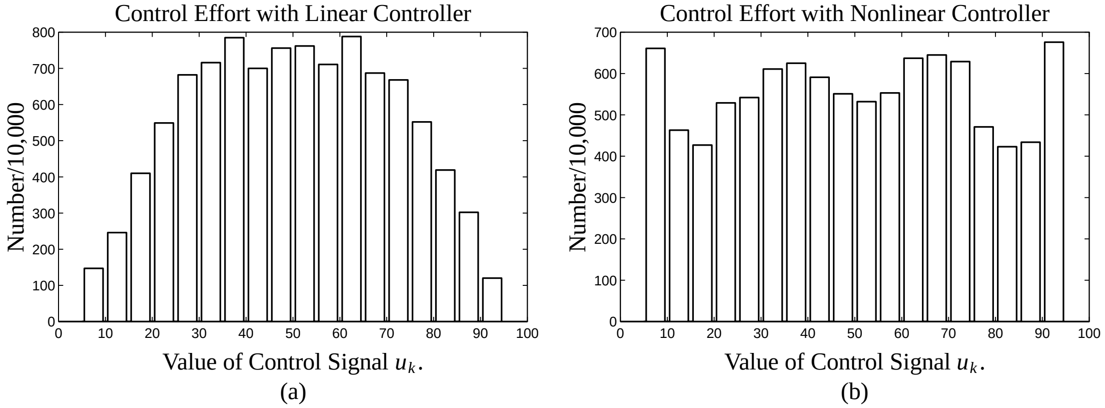


Figure 4.5: Histograms of control effort (for the same command input) when the controller is either linear FIR or nonlinear. The control effort was constrained to be between 5°C and 95°C. The nonlinear controller is better able to take advantage of the full range of allowed control effort. The nonlinear MSE was about 0.67 and the linear MSE was about 1.0.

First, a simple logical argument based on Eq. (4.1) will show that increasing delay will result in decreasing mean-squared system error. If the delay is equal to $-\infty$ then the output of the controller will be zero. This is the worst-case output. If the delay is $+\infty$, then the controller will be a delayed version of the Wiener solution, which is the best-case output. For intermediate delays, consider what happens when the delay goes from $\Delta - 1$ to Δ. One solution for the controller is that all the weights from $C^{(\Delta-1)}$ be copied to $C^{(\Delta)}$, but shifted over one time step. If this is done, the output error will be identical. Therefore, by increasing the delay, we can do *at least as well* as before, but possibly better. Therefore, the mean-squared system error is a non-increasing function of the delay.

Secondly, we need to show that the control effort is an increasing function of the delay. We do this by considering the gain of the controller on the r_k signal. Figure 4.6 shows a comparison of two optimal linear causal controllers (with no constraint on the control effort). One has been designed for a delay of $\Delta - 1$, and the other has been designed for a delay of Δ. They share part of their transfer function up until the node marked ①. From that point on, the transfer functions differ. Therefore, in considering the relative gain of the two controllers, we need only look at the gain from the node marked ① to the two outputs.

The gain of a linear MIMO system, relating maximum input to maximum output, is equal to [47]

$$g(C) \triangleq \sup_{r_k} \frac{\|u_k\|_\infty}{\|r_k\|_\infty} = \max_{i \in [1, N_i]} \sum_{k=0}^{\infty} \sum_{j=1}^{N_o} |c_{k,ij}|,$$

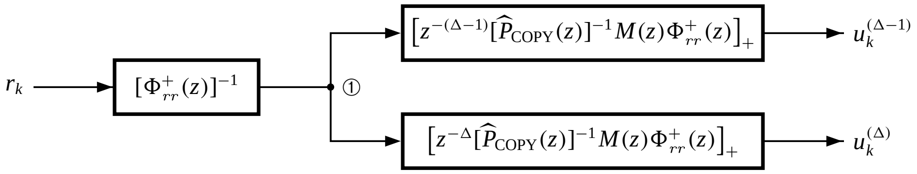

r_k
$[\Phi_{rr}^{+}(z)]^{-1}$
①
$\big[z^{-(\Delta-1)}[\widehat{P}_{\text{COPY}}(z)]^{-1}M(z)\Phi_{rr}^{+}(z)\big]_{+}$
$u_k^{(\Delta-1)}$
$\big[z^{-\Delta}[\widehat{P}_{\text{COPY}}(z)]^{-1}M(z)\Phi_{rr}^{+}(z)\big]_{+}$
$u_k^{(\Delta)}$

Figure 4.6: Two Shannon-Bode optimal controllers: One for a delay of $\Delta - 1$, one for a delay of Δ.

where, N_i is the number of outputs from the controller (inputs to the plant), N_o is the number of inputs to the controller (outputs from the plant), and $c_{k,ij}$ is the kth value of the impulse response relating the jth input to the ith output. In other words, the gain from the input vector to one output is the absolute sum of the impulse responses from all components of the input vector to that output. The maximum of all of these gains is equal to the gain of the system.

The gain of a controller designed to give a system delay of $\Delta - 1$ is compared to the gain of a controller designed for a system delay of Δ

$$\begin{aligned}
\frac{g\big(C^{(\Delta-1)}\big)}{g\big(C^{(\Delta)}\big)} &= \frac{g\Big(\Big[z^{-(\Delta-1)}\big(\widehat{P}_{\text{COPY}}(z)\big)^{-1}M(z)\Phi_{rr}^{+}(z)\Big]_{+}\Big)}{g\Big(\Big[z^{-\Delta}\big(\widehat{P}_{\text{COPY}}(z)\big)^{-1}M(z)\Phi_{rr}^{+}(z)\Big]_{+}\Big)} \\
&\le \frac{g\Big(\Big[z^{-\Delta}\big(\widehat{P}_{\text{COPY}}(z)\big)^{-1}M(z)\Phi_{rr}^{+}(z)\Big]_{+}\Big)}{g\Big(\Big[z^{-\Delta}\big(\widehat{P}_{\text{COPY}}(z)\big)^{-1}M(z)\Phi_{rr}^{+}(z)\Big]_{+}\Big)} \\
&= 1.
\end{aligned}$$

The inequality in the second line is due to the fact that the impulse responses for the shorter delay are identical to the impulse responses for the larger delay, except shifted over one time step and with some taps set to zero due to the causality restriction. Since some taps are set to zero, the absolute sum of the impulses will be lower, and the gain will be lower.

Therefore, we have seen that increasing delay causes decreasing system mean-squared error, but increasing control effort. If control effort is limited, then the controller will no longer give the optimal mean-squared error, and increasing delay causes no further decrease in system mean-squared error.

This is demonstrated using simulations of the nonminimum-phase tank of Sec. 3.2.1. A suite of simulations were performed to find the optimal controller for delays of two through fifteen. A further suite of simulations were performed to find the optimal constrained linear controller for the same delays. A final suite was performed to find the optimal constrained nonlinear controller for the same delays. The unconstrained solution shows progressively lower mean-squared error as the delay increases. However, after a delay of four, the linear constrained solution stops improving. After a delay of six, the nonlinear constrained solution stops improving. Therefore, we might consider using a delay of four if we are satisfied with the linear controller, or a delay of six if we prefer the nonlinear controller.

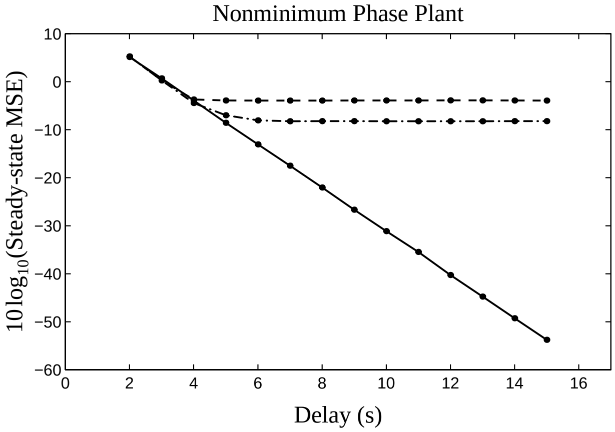


Figure 4.7: Steady-state system mean-squared error versus the system delay. The solid line shows that the steady-state mean-squared error decreases logarithmically if the control effort is unconstrained. The dashed line shows that if the control effort is constrained such that $5°C \le u_k \le 95°C$, and the controller is linear, then the steady-state mean-squared error decreases to a constant value. The dash-dot line is similarly constrained but the controller is nonlinear.

4.3 Synthesis of the Constrained Controller via the BPTM Algorithm

We now begin the second part of this chapter. We have seen some analytic results pertaining to the design of a constrained controller, and are ready to present an algorithm which trains a controller to perform constrained model-reference based control of a linear or nonlinear, SISO or MIMO plant. A key hurdle which must be overcome by the algorithm is to find a mechanism for converting the system error to an adaptation signal used to adjust C. For the most general MIMO nonlinear case, we need some functional block which uses the system error and some form of plant state information to compute the controller error. This block is denoted as “?” in Fig. 4.8.

This functional block must describe an algorithm which also satisfies the following design criteria:

- The algorithm must work with SISO and MIMO, linear and nonlinear plants.
- The algorithm must not be biased by disturbances.

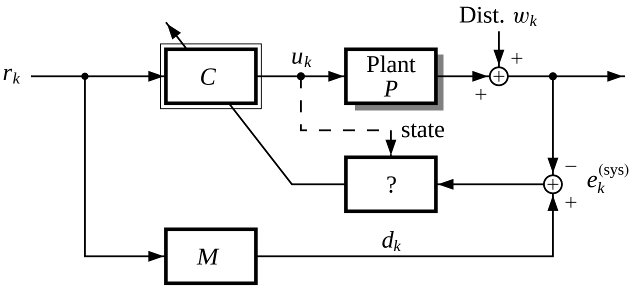


Figure 4.8: Conceptual block diagram of a system used to train the controller C.

- The algorithm must work for autoregressive implementations of $\widehat{P}$ and C.
- The algorithm must minimize some cost functional of the system error *and* the control effort.

The algorithm is now developed.

A related method was introduced in reference [2] where it was referred to as a type of real-time recurrent learning (RTRL) [46]. Here, it is extended in a number of important ways: (1) It is generalized from the SISO case to the MIMO case, where it able to attribute different performance objectives to each plant output; (2) It has been extended from the nonlinear-only case to also incorporate control of linear systems, and (3) It now handles very general (differentiable) constraints on the control signal u_k. We will also see in Chap. 5 that this algorithm can be used to adapt a filter to perform disturbance cancelling.

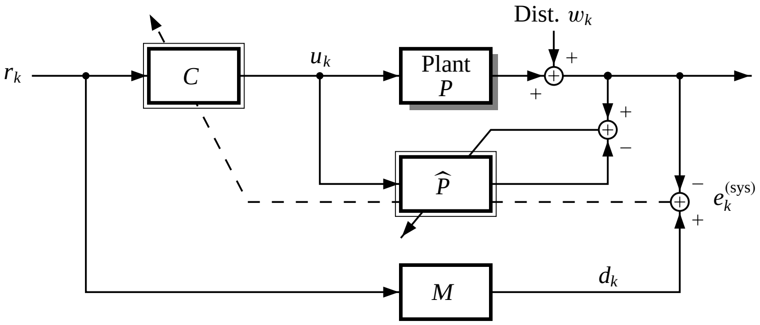


Figure 4.9: Structure diagram illustrating the BPTM method.

Figure 4.9 shows the general framework to be used. Rather than manipulating the block diagram to generate an indirect error signal with which to adapt C, the system error signal is used directly. It is *back-propagated* through the plant model, and there used to adapt the controller. For this reason, the algorithm is named “BackProp Through (Plant) Model” (BPTM).

The algorithm is derived as follows. We wish to train the controller C to minimize the squared system error over a certain trajectory and to simultaneously minimize some function of the control effort. To do so, the system is run for K time steps. At the end of the K time steps, the following sum is computed

$$J_K = \sum_{j=0}^{K} \left\{ e_j^{(\text{sys})^T} Q e_j^{(\text{sys})} + h\big(u_j, u_{j-1}, \ldots, u_{j-r}\big) \right\}.$$

The differentiable function $h(\cdot)$ defines the cost function associated directly with the control signal u_k, and is used to penalize excessive control effort, slew rate and so forth. The system error is the signal $e_k^{(\text{sys})} = d_k - y_k$, and the symmetric matrix Q is a weighting matrix which assigns different performance objectives to each plant output. To minimize the system error over a trajectory of length K and simultaneously minimize a function of the control effort, we must minimize the function J_K.

It is possible to compute the required equations to minimize J_K using gradient descent (in fact, they do not differ very much from what follows). One problem with this approach is that it does not adapt the controller weights in real time. Time is divided into epochs of K time samples in length, and adaptation of the controller weights is performed once at the end of each epoch. For this work, a real-time approach was preferred. Therefore, the same trick is employed as used in [46]. The cost metric J_K is stochastically approximated at each time step as

$$J_k = \left\{ e_k^{(\text{sys})^T} Q e_k^{(\text{sys})} + h(u_k, u_{k-1}, \ldots, u_{k-r}) \right\}.$$

The gradients of the approximate cost function J_k are not the same as the gradients found for the true cost function J_K. Therefore, adaptation is a “noisy” process. In practice, however, it works well.

Continuing, if we let $g(\cdot)$ be the function implemented by the controller C, and $f(\cdot)$ be the function implemented by the plant model $\widehat{P}$, we can state without loss of generality

$$\begin{aligned} u_k &= g\big(u_{k-1}, u_{k-2}, \ldots, u_{k-m}, r_k, r_{k-1}, \ldots, r_{k-q}, W\big) \\ y_k &= f\big(y_{k-1}, y_{k-2}, \ldots, y_{k-n}, u_k, u_{k-1}, \ldots, u_{k-p}\big), \end{aligned} \tag{4.4}$$

where W are the adjustable parameters (weights) of the controller.

As is typical for LMS and backpropagation-like learning methods, the controller weights are updated in the direction of the negative gradient of the cost functional

$$\begin{aligned} \Delta W_k^T &= -\mu \frac{\partial^+}{\partial W} J_k \\ &= -\mu \frac{\partial^+}{\partial W} \left\{ e_k^{(\text{sys})^T} Q e_k^{(\text{sys})} + h(u_k, u_{k-1}, \ldots, u_{k-r}) \right\}, \end{aligned}$$

where μ is the adaptive learning rate. Continuing,

$$\frac{\Delta W_k^T}{\mu} = 2e_k^T Q \left(\frac{\partial^+ y_k}{\partial W} \right) - \left[\sum_{j=0}^{r} \left(\frac{\partial h(u_k, \ldots, u_{k-r})}{\partial u_{k-j}} \right)^T \left(\frac{\partial^+ u_{k-j}}{\partial W} \right) \right]. \tag{4.5}$$

Using Eq. (4.4) and the chain rule for ordered derivatives, two further substitutions may be made at this time

$$\frac{\partial^+ u_k}{\partial W} = \frac{\partial u_k}{\partial W} + \sum_{j=1}^{m} \left(\frac{\partial u_k}{\partial u_{k-j}} \right) \left(\frac{\partial^+ u_{k-j}}{\partial W} \right) \tag{4.6}$$

$$\frac{\partial^+ y_k}{\partial W} = \sum_{j=0}^{p} \left(\frac{\partial y_k}{\partial u_{k-j}} \right) \left(\frac{\partial^+ u_{k-j}}{\partial W} \right) + \sum_{j=1}^{n} \left(\frac{\partial y_k}{\partial y_{k-j}} \right) \left(\frac{\partial^+ y_{k-j}}{\partial W} \right). \tag{4.7}$$

A quick note should be made regarding the dimensions of each term in Eqs. (4.5), (4.6), and (4.7)—if the *plant* has N_i inputs and N_o outputs, and the controller has N_W weights

$$\left[\frac{\partial^+ u_k}{\partial W} \right]_{N_i \times N_W} \quad \left[\frac{\partial^+ y_k}{\partial W} \right]_{N_o \times N_W} \quad \left[\frac{\partial y_k}{\partial y_{k-j}} \right]_{N_o \times N_o}$$

$$\left[\frac{\partial h(\cdot)}{\partial u_{k-j}} \right]_{N_i \times 1} \quad \left[\frac{\partial y_k}{\partial u_{k-j}} \right]_{N_o \times N_i} \quad \left[\frac{\partial u_k}{\partial u_{k-j}} \right]_{N_i \times N_i}$$

Also,

$$[Q]_{N_o \times N_o} \qquad [e_k]_{N_o \times 1} \qquad [u_k]_{N_i \times 1} \qquad [y_k]_{N_o \times 1} \qquad [W]_{N_W \times 1}\,.$$

4.3.1 Linear FIR Plant Model, Linear FIR Controller

The equations for the weight update in their general form, as presented so far, are equally applicable to linear or nonlinear systems. From this point on, the derivations diverge in order to specialize to two different architectures. Here, we look at the example where both the plant model and the controller are linear systems.

Any stable linear plant and linear controller may be approximated with arbitrary precision by FIR filters. Moreover, the feedback terms in $f(\cdot)$ and $g(\cdot)$ are unnecessary and can be dangerous as they invite instability.[1] Therefore, we assume that both the the plant model and the controller are FIR.

The input to the controller filter is a tapped-delay-line of $q+1$ vectors, each of length N_o. We define this composite vector, at time k, to be

$$R_k \triangleq \left[r_k^T \; r_{k-1}^T \; \cdots \; r_{k-q}^T \right]^T.$$

[1] Even if the plant model, for example, is not implemented as an FIR filter, an FIR impulse response can be generated by using the plant model to filter an impulse signal. Therefore, the results of this section are more general than they appear.

The plant has N_i inputs. Therefore, the controller will have N_i different linear filters operating on R_k to produce the control signal u_k. We let W_1^T be the first such filter, W_2^T be the second, and so on. We may organize these filters into a *matrix* $\mathbf{W}_c$ such that

$$\mathbf{W}_c = \left[W_1 \; W_2 \; \cdots W_{N_i}\right]^T.$$

Then,

$$u_k = \mathbf{W}_c R_k. \tag{4.8}$$

This definition is useful for actually computing u_k. However, for the purpose of adapting all the weight values of the controller it is also useful to define the *column vector* of weights to be

$$W = \left[W_1^T \; W_2^T \; \cdots W_{N_i}^T\right]^T. \tag{4.9}$$

Note that $\mathbf{W}_c$ is a matrix while W is a vector; both contain identical information in different arrangements. One is a digital copy and rearrangement of the other. We wish to adapt the values in W to optimize J_k.

The input to the plant model is analogous to the input to the controller—it is a tapped-delay-line of $p+1$ vectors, each of length N_i. We define this composite vector, at time k, to be

$$U_k \triangleq \left[u_k^T \; u_{k-1}^T \; \cdots \; u_{k-p}^T\right]^T.$$

In further likeness to the controller, the plant model has weight matrix $\mathbf{W}_{\widehat{p}}$ which acts the same way as the controller weight matrix $\mathbf{W}_c$. So,

$$y_k = \mathbf{W}_{\widehat{p}} U_k.$$

With these definitions, we may solve Eq. (4.5) to find the weight update. We need to find three quantities: $\partial h(\cdot)/\partial u_{k-j}$, $\partial^+ u_k/\partial W$ and $\partial^+ y_k/\partial W$.

First, we note that $\partial h(\cdot)/\partial u_{k-j}$ depends on the user-specified function $h(\cdot)$. It can be calculated given $h(\cdot)$. Later, we will see that it is useful to arrange the result in a composite vector, defined at time k to be

$$dH_k \triangleq \left[\left(\frac{\partial h(\cdot)}{\partial u_k}\right)^T \left(\frac{\partial h(\cdot)}{\partial u_{k-1}}\right)^T \cdots \left(\frac{\partial h(\cdot)}{\partial u_{k-r}}\right)^T\right]^T. \tag{4.10}$$

Secondly, we consider $\partial^+ u_k/\partial W$ as expanded in Eq. (4.6). Since the controller is assumed to be FIR, this simplifies

$$\frac{\partial^+ u_k}{\partial W} = \frac{\partial u_k}{\partial W}.$$

From the definition of u_k in Eq. (4.8), and the ordering of W in Eq. (4.9), we see $\partial u_k/\partial W$ is a block-diagonal matrix containing N_i copies of R_k^T

$$\frac{\partial u_k}{\partial W} = \operatorname{diag}\left\{R_k^T, R_k^T, \cdots, R_k^T\right\} = \begin{bmatrix} R_k^T & 0 & \dots & 0 \\ 0 & R_k^T & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R_k^T \end{bmatrix}.$$

Thirdly, we consider $\partial^+ y_k/\partial W$, as expanded in Eq. (4.7). Since the plant model is assumed to be FIR, we may re-write the expression as

$$\frac{\partial^+ y_k}{\partial W} = \sum_{j=0}^{p} \left(\frac{\partial y_k}{\partial u_{k-j}}\right)\left(\frac{\partial^+ u_{k-j}}{\partial W}\right). \tag{4.11}$$

The first term in this summation, $\partial y_k/\partial u_{k-j}$, is equal to the N_i columns of $\mathbf{W}_{\widehat{p}}$ associated with the input u_{k-j}. The second term is $\partial^+ u_k/\partial W$, as calculated for this time-step and the previous p time steps. We have seen how to calculate this. We can put all of this together by defining

$$dU_k \triangleq \left[\left(\frac{\partial u_k}{\partial W}\right)^T \left(\frac{\partial u_{k-1}}{\partial W}\right)^T \cdots \left(\frac{\partial u_{k-p}}{\partial W}\right)^T\right]^T. \tag{4.12}$$

Then Eq. (4.11) can be computed to be

$$\frac{\partial^+ y_k}{\partial W} = \mathbf{W}_{\widehat{p}}\; dU_k.$$

Finally, we combine all the above to get

$$\Delta W_k = \mu\Big(\big[2e_k^T Q \mathbf{W}_{\widehat{p}} - dH_k^T\big]\, dU_k\Big)^T. \tag{4.13}$$

The curious reader may wish to verify the result of Eq. (4.13) by considering it in its simplest form. Suppose that the plant is SISO and has no dynamics (*i.e.*, $\mathbf{W}_{\widehat{p}} = [1]$). Furthermore, suppose that $h(\cdot) \equiv 0$ and that $Q = 1$. Then, the adaptation rule reduces to the regular LMS rule for adapting a filter. If, however, the plant has some dynamics,

$$\Delta W_k = 2\mu e_k \left(\sum_{j=0}^{p} \mathbf{W}_{\widehat{p},j} R_{k-j}\right),$$

which is the LMS rule weighted by the impulse response of the plant. Since the plant is assumed stable, $\mathbf{W}_{\widehat{p},j} \to 0$ as j gets large, and the current inputs are weighted more strongly than past inputs.

The bottom line: Figure 4.10 shows a summary of BPTM for linear SISO or MIMO plants. Ultimately, the weight update is calculated using Eq. (4.13). Prior known quantities are: μ, and Q. The error $e_k^{(\text{sys})}$ is an input to the weight-update process, and the weights of the plant model $\mathbf{W}_{\widehat{p}}$ are also known. Thus, it remains to compute dH_k and dU_k. The dH_k vector is a function of the vector U_k, which is known as it is the input to the plant model. Similarly, dU_k is a function of dU_{k-1} and the vector R_k, which is known as it is the input to the controller. Notice that the algorithm is extremely simple. No computations other than matrix multiplies are necessary. The algorithm may be implemented in one line of *Matlab* code, for example.

```
begin {Adapt C (linear)}
    Initialize $dU_k$ to $\mathbf{0}$.
    Compute $dH_k$ as a function of $U_k$.
    Shift $dU_k$ down $N_i$ rows, and shift in $N_i$ copies of $R_k$.
    Compute weight update according to Eq. (4.13).
end {Adapt C (linear)}
```

Figure 4.10: An algorithm for adapting C when the controller and plant model are linear. The first step is performed once, and the other steps are performed in sequence, iteratively as time progresses.

4.3.2 Nonlinear NARX Plant Model, Nonlinear NARX Controller

Given the background material in Chap. 2, the derivation of BPTM for a nonlinear plant and nonlinear controller is actually somewhat more compact than for the linear case. This is true even though autoregressive plant models and controllers are generally required for nonlinear control. The dynamical behavior of most nonlinear systems may not be well approximated by a nonlinear transversal model.

Whereas for the linear plant we had some freedom to choose the structure of the plant model, here we need to restrict ourselves to a single paradigm. This is because there is no equivalent in the nonlinear domain to an “impulse response.” The BPTM algorithm, for a linear plant, was able to compute the impulse response of the plant model, regardless of the structure of the model, and use that to update the controller. Here, we assume that NARX neural network filters are used for both the plant model and the controller. Such filters are universal function approximators and are capable of controlling any (controllable) nonlinear system with acceptable accuracy.

To restate the problem at hand, we desire to compute the weight-update ΔW_k of Eq. (4.5). The only terms which differ from the linear derivation are those of Eqs. (4.6) and (4.7), which are repeated here for convenience

$$\frac{\partial^+ u_k}{\partial W} = \frac{\partial u_k}{\partial W} + \sum_{j=1}^{m} \left(\frac{\partial u_k}{\partial u_{k-j}}\right)\left(\frac{\partial^+ u_{k-j}}{\partial W}\right) \tag{4.6}$$

$$\frac{\partial^+ y_k}{\partial W} = \sum_{j=0}^{p} \left(\frac{\partial y_k}{\partial u_{k-j}}\right)\left(\frac{\partial^+ u_{k-j}}{\partial W}\right) + \sum_{j=1}^{n} \left(\frac{\partial y_k}{\partial y_{k-j}}\right)\left(\frac{\partial^+ y_{k-j}}{\partial W}\right). \tag{4.7}$$

A quick examination of these equations will be adequate to see that the terms which need to be computed at each iteration are

$$\frac{\partial u_k}{\partial W}, \qquad \frac{\partial u_k}{\partial u_{k-j}}, \qquad \frac{\partial y_k}{\partial u_{k-j}}, \quad \text{and} \quad \frac{\partial y_k}{\partial y_{k-j}}.$$

The first term is the direct effect of the controller weights on the controller output, and the other terms are the effects of the inputs of the controller and plant model to their respective outputs. They are all Jacobian matrices and are very simple to calculate for any neural network, using the backpropagation algorithm (as described in Sec. 2.3.2). Armed with this information, we may readily compute the required terms of Eqs. (4.6) and (4.7). All that is required is a little careful bookkeeping.

The $\partial^+ u_k/\partial W$ term of Eq. (4.6) is computed by determining the values of $\partial u_k/\partial W$ and $\partial u_k/\partial u_{k-j}$. These are found by back-propagating unit vectors $v = \hat{e}_i$ through the controller neural network and using Eqs. (2.3) and (2.4) as appropriate.

The $\partial^+ y_k/\partial W$ term of Eq. (4.7) is then computed. To do so, we need to know $\partial y_k/\partial u_{k-j}$ and $\partial y_k/\partial y_{k-j}$, which are found by back-propagating unit vectors $v = \hat{e}_i$ through the plant-model neural network and using Eq. (2.3).

A practical implementation is realized by compacting the notation into a collection of matrices as before. The definitions of dU_k and dH_k remain unchanged from Eqs. (4.12) and (4.10). Furthermore, we define

$$dY_k \triangleq \left[\left(\frac{\partial y_{k-1}}{\partial W}\right)^T \left(\frac{\partial y_{k-2}}{\partial W}\right)^T \cdots \left(\frac{\partial y_{k-n}}{\partial W}\right)^T \right]^T$$

$$\partial_U Y_k \triangleq \left[\left(\frac{\partial y_k}{\partial u_k}\right)^T \left(\frac{\partial y_k}{\partial u_{k-1}}\right)^T \cdots \left(\frac{\partial y_k}{\partial u_{k-p}}\right)^T \right]^T$$

$$\partial_Y Y_k \triangleq \left[\left(\frac{\partial y_k}{\partial y_{k-1}}\right)^T \left(\frac{\partial y_k}{\partial y_{k-2}}\right)^T \cdots \left(\frac{\partial y_k}{\partial y_{k-n}}\right)^T \right]^T$$

$$\partial_U U_k \triangleq \left[\left(\frac{\partial u_k}{\partial u_{k-1}}\right)^T \left(\frac{\partial u_k}{\partial u_{k-2}}\right)^T \cdots \left(\frac{\partial u_k}{\partial u_{k-p}}\right)^T \right]^T.$$

```
begin {Adapt $C$ (nonlinear)}
    Update $\partial^+ u_k/\partial W$:
        • Shift $dU_k$ down $N_i$ rows.
        • Backpropagate $N_i$ unit vectors through $C$ to form $\partial_U U_k$ and $\partial u_k/\partial W$. Each backpropagation produces one row of both matrices.
        • Compute top $N_i$ rows of $dU_k$ to be $\partial u_k/\partial W + (\partial_U U_k)(dU_k)$.
    Update $\partial^+ y_k/\partial W$:
        • Backpropagate $N_o$ unit vectors through $\widehat{P}$ to form $\partial_U Y_k$ and $\partial_Y Y_k$. Each backpropagation produces one row of both matrices.
        • Compute $d_W Y_k = (\partial_U Y_k)(dU_k) + (\partial_Y Y_k)(dY_k)$.
        • Shift $dY_k$ down $N_o$ rows and save $d_W Y_k$ in the top $N_o$ rows.
    Compute $dH_k$.
    Update Weights:
        • Compute $\Delta W_k^T = 2\mu e_k^T Q(d_W Y_k) - \mu(dH_k^T)(dU_k)$.
        • Adapt, enumerating weights in the same order as when computing $\partial_W U_k$.
end {Adapt $C$ (nonlinear)}
```

Figure 4.11: Algorithm to adapt a NARX controller for a NARX plant model.

The bottom line: The algorithm to adapt a NARX controller for a NARX plant model is summarized in Fig. 4.11. Any programming language supporting matrix mathematics can very easily implement this algorithm. It works well.

4.3.3 Separability for Efficient Implementation

The preceding discussion introduced an algorithm to train an adaptive controller to control a plant. Two specific examples were derived in detail: (1) A linear FIR controller for a linear FIR plant model; (2) A NARX neural network controller for a NARX neural-network plant model. But, what if we want to adapt a NARX controller for a linear plant? (see, for example, Sec. 4.4.1) What if we want a nonlinear transversal filter plant model or controller? (see, for example, Sec. 4.4.5) It would seem that if there are N possible options for the structure of either the adaptive controller or the adaptive plant model, then our suite of algorithms must have N^2 subroutines to be able to handle all combinations. Each time we add a new structure to our repetoire, we move from N to $N+1$ types and need to add $2N+1$ algorithms to our collection![2]

Fortunately, this is not the case. A quick examination of Eqs. (4.6) and (4.7) verifies this fact. We see that $\partial^+ u_k/\partial W$ depends only on information local to the controller. The term $\partial^+ y_k/\partial W$ depends only on information local to the plant model as well as the value of $\partial^+ u_k/\partial W$. This data-flow relationship is depicted in Fig. 4.12.

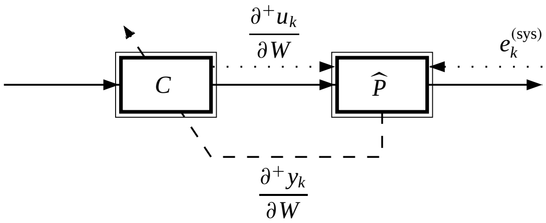

$\frac{\partial^+ u_k}{\partial W}$
$e_k^{(\text{sys})}$
C
$\widehat{P}$
$\frac{\partial^+ y_k}{\partial W}$

Figure 4.12: Data-flow diagram illustrating the algorithmic independence of BPTM on the structures of C and $\widehat{P}$.

Therefore, each type of structure needs only two subroutines. One subroutine computes $\partial^+ u_k/\partial W$ and the other computes $\partial^+ y_k/\partial W$ given $\partial^+ u_k/\partial W$ and $e_k^{(\text{sys})}$. If we have N candidate structures, our control suite needs only $2N$ algorithms. Adding a new structure means that we need to add only two subroutines. This is a very important feature of the algorithm which makes it feasible as a control technology.

4.3.4 Initialization of Linear Controllers

The BPTM learning method, when used to control linear plants, will train a controller regardless of the initial weights of the controller. If the plant model is initialized to some non-zero function, the weights of the controller may even start at zero.

The size of the controller is chosen by the designer to balance the trade-off between training time and performance. In a low-cost-of-control scenario, the number of weights required by the

[2] For example, in this dissertation, structures used for the plant model were: linear FIR, linear IIR (connected in parallel), linear IIR (connected in series-parallel), nonlinear transversal filters, NARX filters (connected in parallel) and NARX filters (connected in series-parallel). Structures used for the controller and disturbance canceller were: linear FIR, linear IIR (connected in parallel), nonlinear transversal filters, and NARX filters (connected in parallel). Instead of needing $7 \times 4 = 28$ algorithms, only $7 + 4 = 11$ were needed.

controller may be quite large. This, however, greatly increases the training time of the system. What is desired is a clever way to initialize the weights so that they are very nearly optimal before the training commences. This section outlines a method to do this for a linear controller. It is an extension of the system-identification method called "empirical transfer function estimate."

If there are no constraints on the control effort, the weights of the controller are initialized to an estimate of the Wiener solution

$$C(e^{j\omega}) = \big(\widehat{P}(e^{j\omega})\big)^{-1}\big(M(e^{j\omega})\big). \tag{4.14}$$

This is computed by taking the FFT of the reference model impulse response and the plant model impulse response and dividing them at each frequency. After initialization, the controller is adapted using the BPTM algorithm to improve the solution. Further adaptation is required for a number of reasons

- Equation (4.14) is the Wiener solution for an unconstrained controller. We really desire the Shannon-Bode solution. This may be computed instead of Eq. (4.14) but requires knowledge of the statistics of the input signal r_k, and factoring of its spectrum. Using the Wiener solution is fine for a minimum-phase plant, and is pretty good for a nonminimum-phase plant if the delay in $M(z)$ is large enough.

- The expression being used is the Wiener solution for an IIR controller. Our controllers are FIR. Simply truncating the response does not give the Wiener solution for an FIR controller.

- The implementation is done using the FFT algorithm. Hence, each frequency is not being matched, as the equation would suggest, but only discrete frequencies within the band of interest. This problem may be largely mitigated by zero-padding the time-domain signals before taking the FFT (A good rule of thumb seems to be to zero-pad the signals to eight times their original length).

- The expression $\big(P(e^{j\omega})\big)^{-1}\big(M(e^{j\omega})\big)$ is not being implemented; rather, we use the plant model and compute $\big(\widehat{P}(e^{j\omega})\big)^{-1}\big(M(e^{j\omega})\big)$. If the plant model is significantly different from the plant, or if the model is corrupted by noise, the solution will be poor. Some sort of Wiener-inspired filtering can be performed on $\widehat{P}(e^{j\omega})$ to try to eliminate the effects of noise. For example:[3]

$$C(e^{j\omega}) = \frac{M(e^{j\omega})}{\widehat{P}(e^{j\omega})}\,\frac{\|\widehat{P}(e^{j\omega})\|^2 + k^2}{\|\widehat{P}(e^{j\omega})\|^2}.$$

[3] This is similar in form to the Wiener-optimal filter which would filter out the effects of white noise.

If there are constraints on the control effort, then Eq. (4.14) may be used to estimate $C_{\text{causal}}^{(\text{opt})}(z)$, which is then used in the algorithm of Fig. 4.3 to compute an initial guess for the controller.

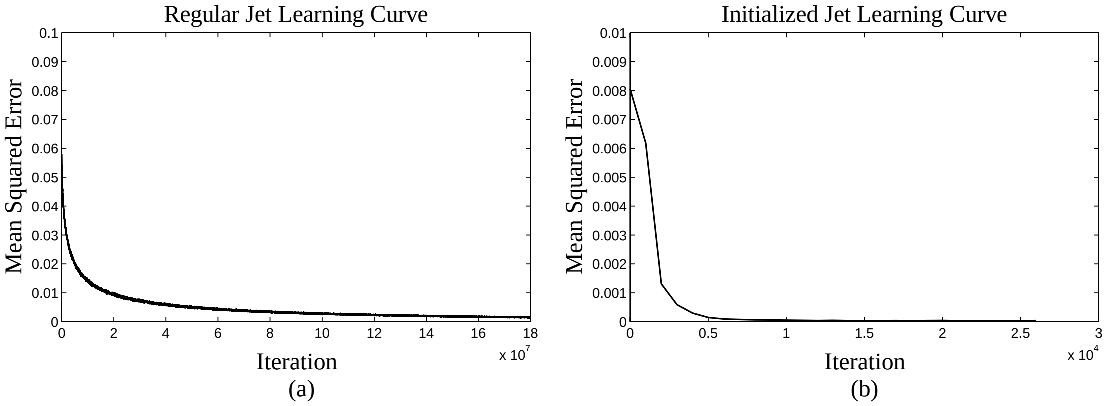


Figure 4.13: Comparison of learning curves when training a controller for the Boeing 747 example of Sec. 3.2.2. (a) Controller initialized to zero; (b) Controller initialized using Eq. (4.14). In both cases the plant model was pre-trained.

Figure 4.13 shows a comparison of the learning curves (for the example of Sec. 3.2.2) when the controller was and was not initialized using Eq. (4.14). When the controller was initialized to zero, training took more than 18×10^7 iterations. When the controller was pre-initialized using Eq. (4.14), a better solution was obtained almost immediately, and the process converged after approximately 1×10^4 iterations. An improvement of over three orders of magnitude can be realized!

4.3.5 Initialization of Nonlinear Controllers

Initialization of a nonlinear controller is much more difficult than for a linear controller. No known analytical solutions are available beyond the expressions in Chap. 2. These equations are not useful for initializing the weights of a nonlinear neural-network filter since they specify the optimal function and not the optimum weights.

To initialize a nonlinear controller we need some prior knowledge of how that controller *should* function. If we have this prior information, we can create a fixed controller which coarsely controls the plant. An adaptive controller is placed in parallel with the fixed controller and is adapted to fine-tune the output of the fixed controller. This control methodology is shown in Fig. 4.14.

This merger of engineering know-how and adaptive fine-tuning presents a practical marriage between the technologies of fuzzy control and adaptive neural control. Fuzzy logic can compactly

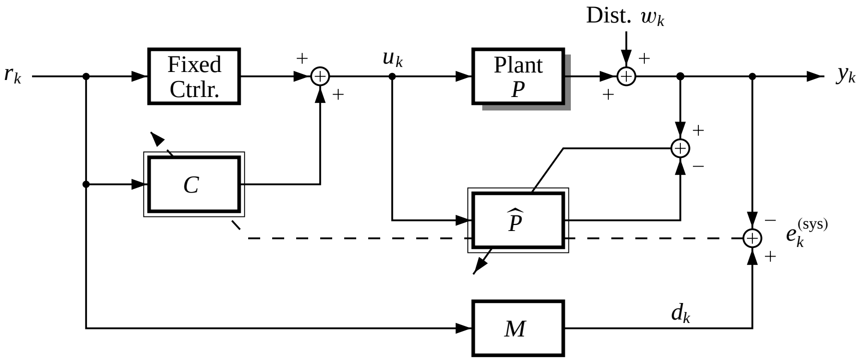


Figure 4.14: Diagram showing a possible method of initializing a nonlinear controller.

represent prior knowledge, and neural networks can fine-tune the result. Much future work needs to be done researching this possibility. No results are available at this time.

4.4 Simulation Examples

We have now seen some analytic results pertaining to constrained control, and an algorithm for adapting a controller to perform constrained control. This final section presents a number of simulation examples to demonstrate the algorithm just developed.

4.4.1 Minimum-Phase Linear SISO Plant

The first examples are for the minimum-phase tank of Sec. 3.2.1. Equation (3.1) is the difference equation specifying the dynamics of the plant. For all simulations, the input source was equal to the constant 50$^\circ C$ plus a first-order Markov process with pole at $z = 0.7$ and driven by i.i.d. uniform random variables with zero mean and maximum magnitude 5$^\circ C$. When constraints on the control effort were considered, the control signal u_k was restricted to be between 5$^\circ C$ and 95$^\circ C$. The controller was a twenty-tap FIR filter.

Figure 4.15 shows the penalty function, $h(u_k)$, used when adapting a constrained controller. The penalty is zero for control effort between 5.5$^\circ C$ and 94.5$^\circ C$. For control effort outside this range, a parabolic function was used to compute the penalty. In Fig. 4.15(a), the overall penalty function is plotted. It appears to be a hard limit on the control signal. However, in Fig. 4.15(b), a region of Fig. 4.15(a) is magnified to show the parabolic nature of the constraint. The actual equation governing the penalty function is:

$$h(u_k) = \begin{cases} \left(\frac{u_k - 5.5}{5 - 5.5}\right)^2, & \text{if } u_k < 5.5; \\ \left(\frac{u_k - 94.5}{95 - 94.5}\right)^2, & \text{if } u_k > 94.5; \\ 0, & \text{otherwise.} \end{cases} \tag{4.15}$$

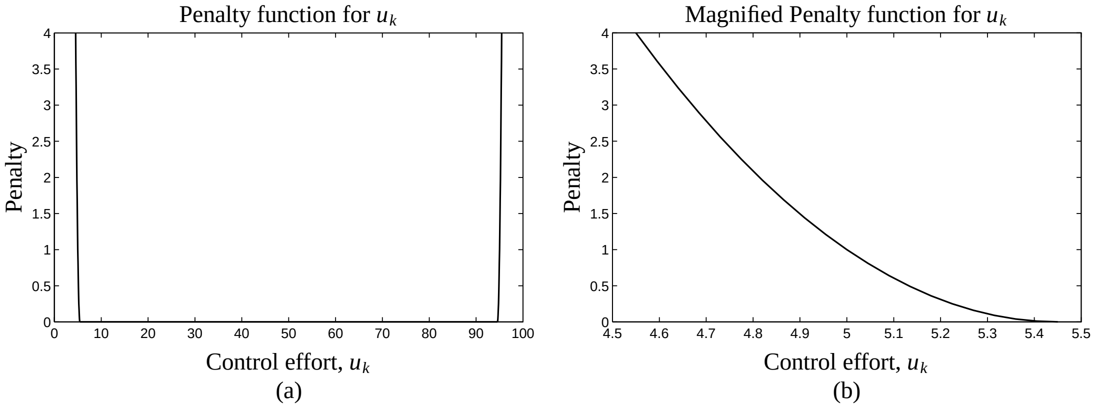
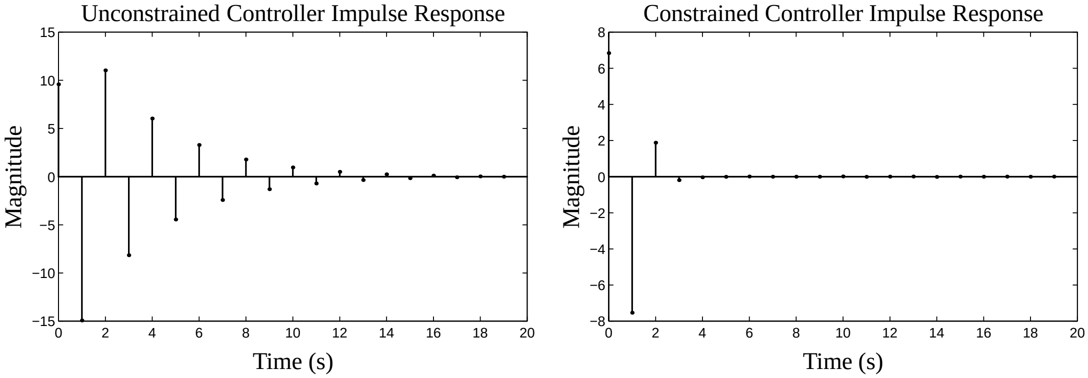


Figure 4.15: Penalty function used on the magnitude of the control effort.

The reference-model was a simple delay of two sample periods: $M(z) = z^{-2}$. This is required for precise control of this plant since it is a generalized minimum-phase plant, and may be perfectly controlled only for delays greater than or equal to two samples. Simulations were performed to determine the controller with and without constraints on the control effort. Figure 4.16 shows the impulse response of both controllers. It is easy to see that the gain (absolute sum of the impulses) of the unconstrained controller is much larger.

Figure 4.17 shows the tracking performance of the unconstrained and constrained controllers for identical first-order Markov reference signals. Two constrained controllers were trained. The first was a linear controller, implemented as an FIR filter with 20 taps. The second was a nonlinear controller, implemented by a $\mathcal{N}_{(4,1):20:1}$ neural controller. In Figs. 4.17(a), (c), and (e), the tracking performance of these three controllers are compared. The solid line shows the continuous-time output of the plant, computed by simulating the continuous-time differential equations governing the plant output. The dotted line shows the discrete-time output of the plant, and the gray stair-step line shows the desired response of the plant output. The desired response exists only at the sampling intervals, but is shown as a continuous stair-step line to make reading the figure easier.

We see that the unconstrained controller (nearly) exactly controls the discrete-time output of the plant. That is, the output of the plant is nearly the same as the desired response *at the sampling*



(a)

(b)

Figure 4.16: Impulse responses of the unconstrained and constrained controllers for the minimum-phase tank.

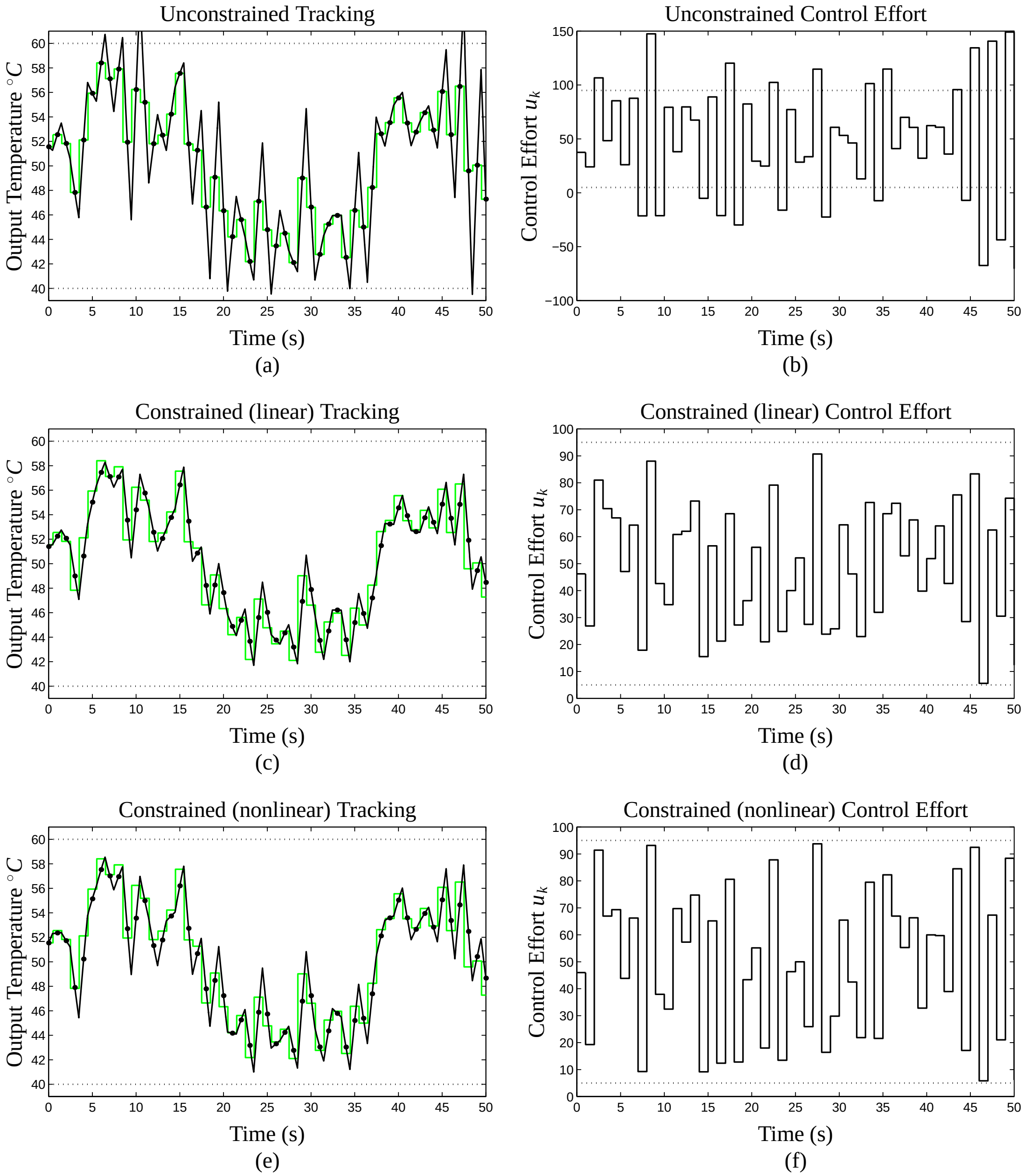
instants. The constrained controllers are not able to match this performance at the sampling instants. However, the inter-sample responses of the constrained controllers are much more reasonable than the response of the unconstrained controller.

Figures 4.17(b), (c) and (f) show the control effort required by the three controllers for the same input signal. We see that the unconstrained controller produces control signals outside the allowed range of $5^\circ C \leq u_k \leq 95^\circ C$. Both of the constrained controllers produce acceptable control signals. We can compare the figures visually and see many similarities. The nonlinear constrained controller makes the best advantage of its allowed range and performs the best, overall. Figure 4.5 also shows that the nonlinear controller is better able to use the allowed range of control effort.

Figure 4.18 is similar to Fig. 4.17, but plots the step response of the system rather than the tracking response. In Fig. 4.18(a), we can very easily see the ringing between samples caused by the excessive control effort demanded by the unconstrained controller. The step responses of the linear and nonlinear controllers are much more reasonable, with the step response of the nonlinear controller being the smoothest, and having the fastest settling time.

4.4.2 Nonminimum-Phase Linear SISO Plant

The second set of examples are for the nonminimum-phase tank of Sec. 3.2.1. Equation (3.2) specifies the dynamics of the plant. The input source was equal to the constant $50^\circ C$ plus a first-order Markov process with pole at $z = 0.7$ and driven by i.i.d. uniform random variables with zero mean and maximum magnitude $5^\circ C$. When constraints on the control effort were considered, the



(a) (b)

(c) (d)

(e) (f)

Figure 4.17: Tracking performance of three controllers for the minimum-phase tank example. In (a), (c), and (e), the tracking performance is shown. The solid line is the continuous-time output of the plant. The dotted line is the plant output at the sample instants. The gray line is the desired response at the sample instants. In (b), (d), and (f), the control effort required is shown.

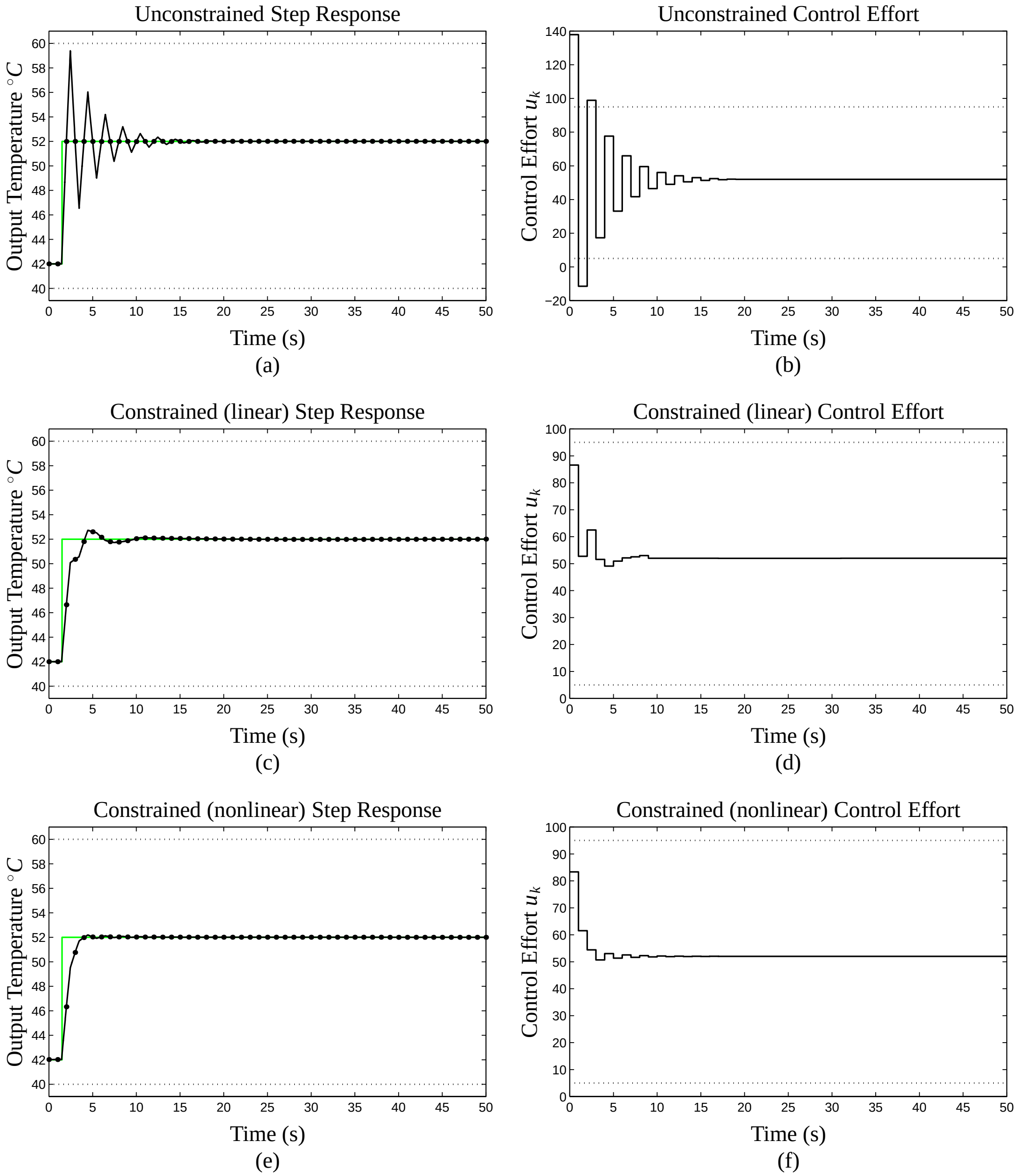


Figure 4.18: Step response of three controllers for the minimum-phase tank example. In (a), (c), and (e), the step response is shown. The solid line is the continuous-time output of the plant. The dotted line is the plant output at the sample instants. The gray line is the desired step. In (b), (d), and (f), the control effort required is shown.

control signal u_k was restricted to be between $5^\circ C$ and $95^\circ C$. Figure 4.15 shows the penalty function $h(u_k)$ used when adapting the constrained controller $\big($see also Eq. (4.15)$\big)$.

The reference-model was a variable delay which took on values between two to fifteen sample periods: $M(z) = z^{-\Delta},\ \Delta \in [2 \ldots 15]$. The different delays caused different levels of performance, as shown in Fig. 4.7. The longer the delay, the smaller the steady-state mean-squared system error.

Simulations were performed to determine the controller with and without constraints on the control effort. Figure 4.20 shows the impulse responses of several of the *controllers* when there were no constraints. It also shows the *system* impulse response. It is easy to see that as the delay increases, the system impulse response approaches a delayed impulse, as it should. Figure 4.19 shows the impulse response for a constrained linear controller, and the resulting system impulse response. Clearly, performance using the constrained controller is worse than if the controller is unconstrained.

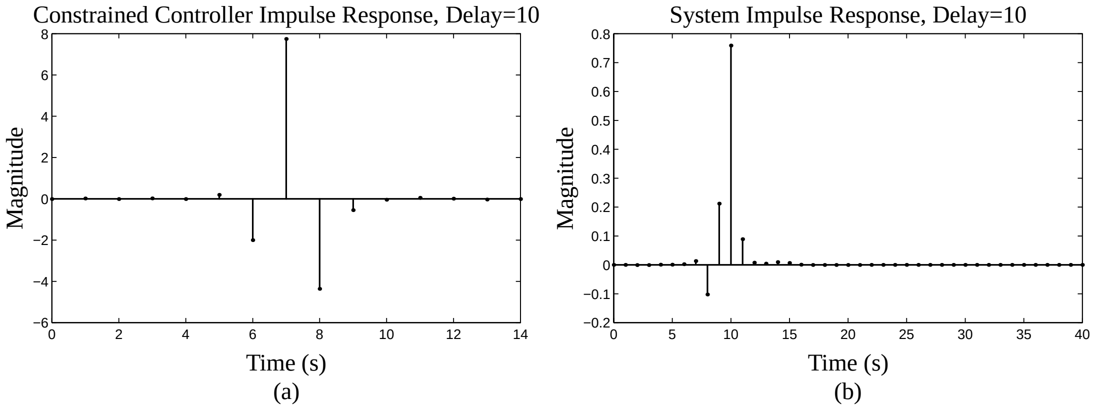
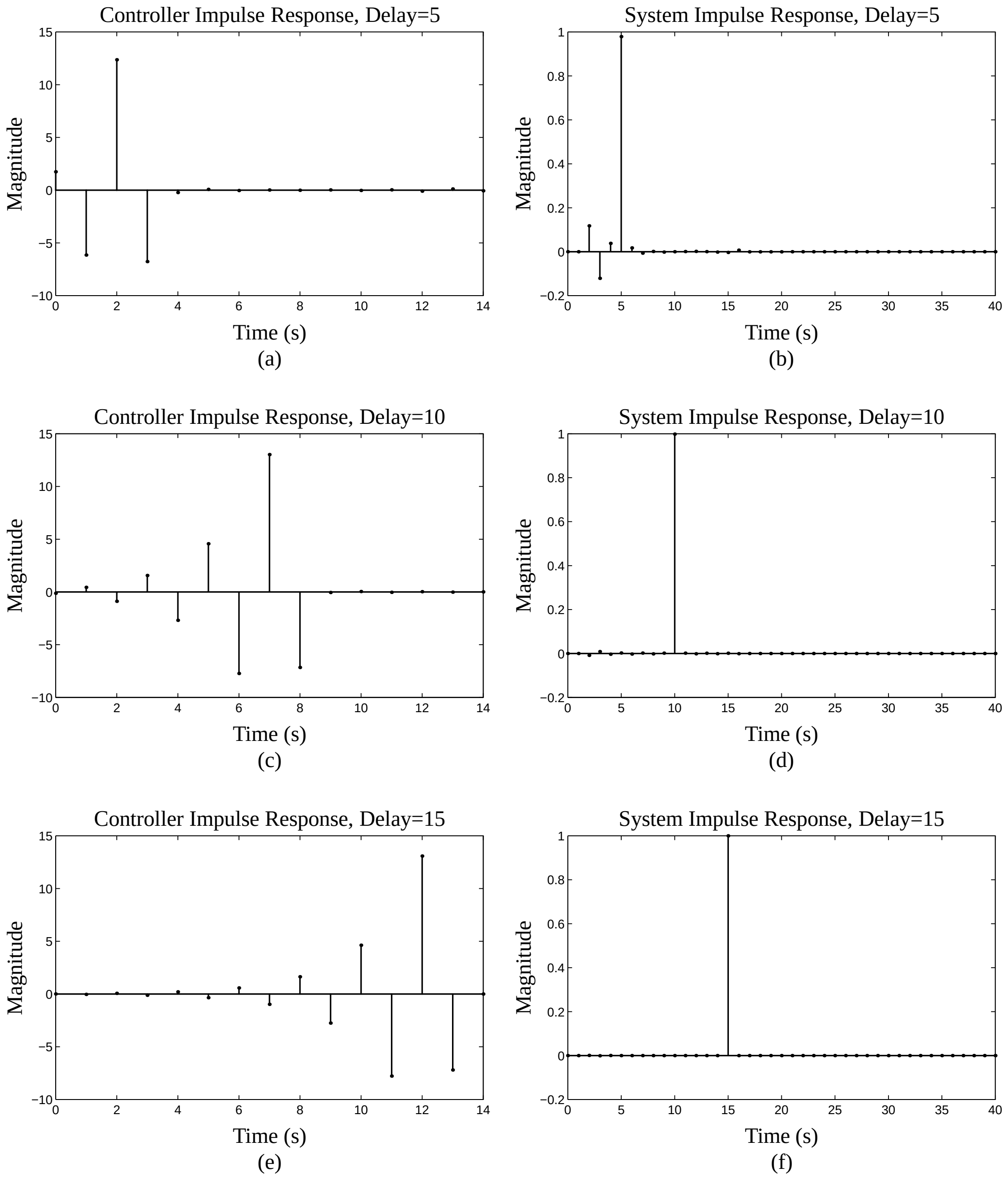


Figure 4.19: Impulse response of the constrained controller for the nonminimum-phase tank.

Tracking performance and step responses were similar to those shown in Figs. 4.17 and 4.18, so are not shown. As with the minimum-phase tank example, Fig. 4.7 shows that a nonlinear controller ($\mathcal{N}_{(20,1):20:1}$) can provide much better performance for constrained control than a linear controller.

4.4.3 Linear MIMO Plant

The third set of simulations demonstrate multi-input multi-output linear control. The problem is to control the lateral motion of a Boeing 747 airliner (from Sec. 3.2.2), linearized around the operating "point" of Mach 0.8 velocity and 40,000 ft. altitude. The inputs to the plant are the rudder and aileron angles (in degrees) and the output of the plant is the yaw rate (in radians/second) and the



Controller Impulse Response, Delay=5

System Impulse Response, Delay=5

Controller Impulse Response, Delay=10

System Impulse Response, Delay=10

Controller Impulse Response, Delay=15

System Impulse Response, Delay=15

(a) (b) (c) (d) (e) (f)

Figure 4.20: Impulse responses of constrained controllers for the nonminimum-phase tank. System delays of 5, 10, and 15 samples are shown. The system impulse response becomes closer to a delayed impulse as the delay increases.

bank angle (in radians). The inputs to the controller are thus the desired yaw rates and bank angles. The desired yaw rate was modeled as a first-order Markov process, with a pole at $z = 0.9$ and an input stream of i.i.d. uniform random variables with zero mean and maximum magnitude 0.03. The desired bank angle was also modeled as a first-order Markov process (independent of the desired yaw rate), with a pole at $z = 0.9$ and an input stream of i.i.d. uniform random variables with zero mean and maximum magnitude 0.12. The plant is minimum-phase so the reference model was a unit delay.

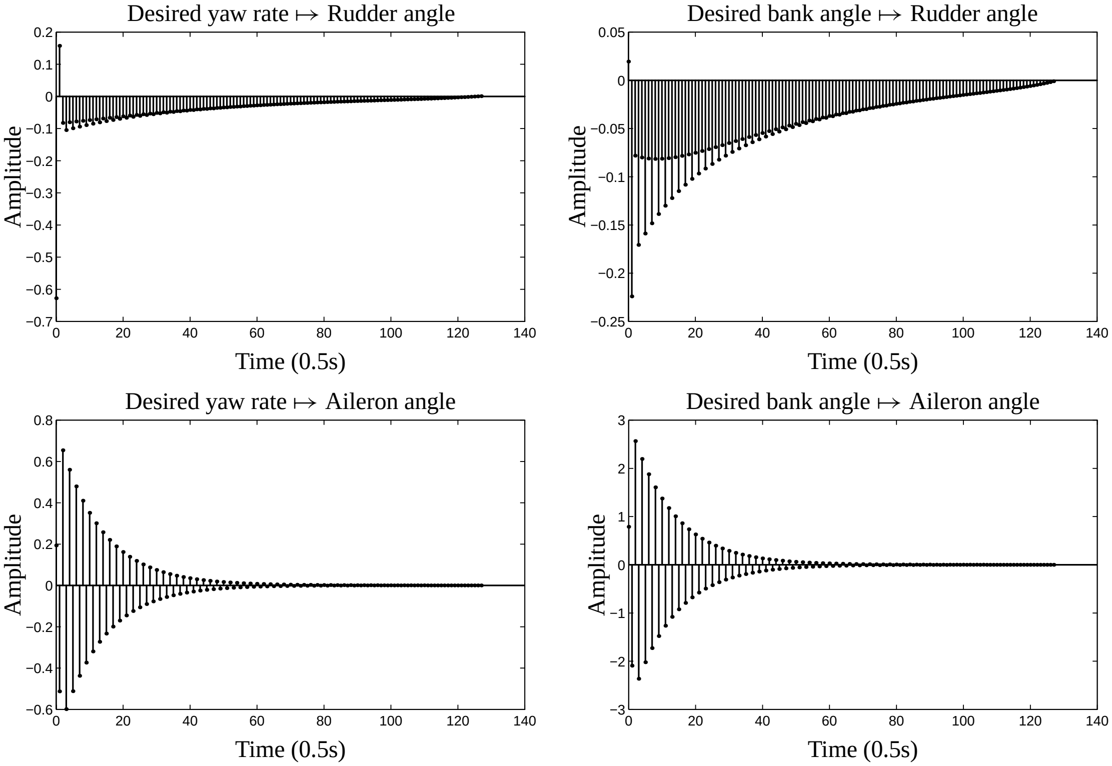


Figure 4.21: The four impulse responses comprising the Boeing 747 controller. The controller had no constraints on the control effort.

Since there are two inputs and two outputs, four transfer functions are used to specify the input-output behavior, and these can be represented as four impulse responses. Figure 4.21 shows the impulse-response matrix for the converged controller. As can be seen, the architecture of the controller was four FIR filters of length 128.

The tracking response and control effort for a characteristic input signal are plotted in Fig. 4.22. In this figure, the desired response is a gray line, the actual discrete-time response is plotted as dots at each sample instant, and the continuous-time resonse is plotted as a solid line. Like the

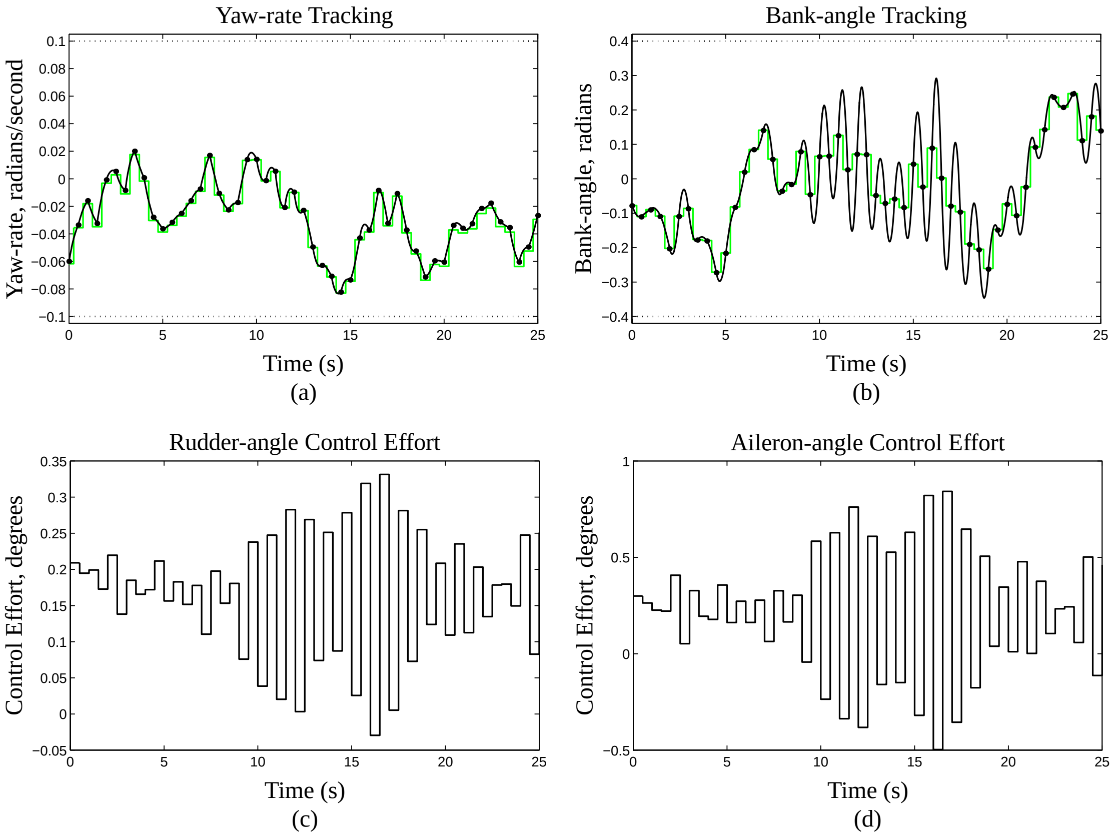
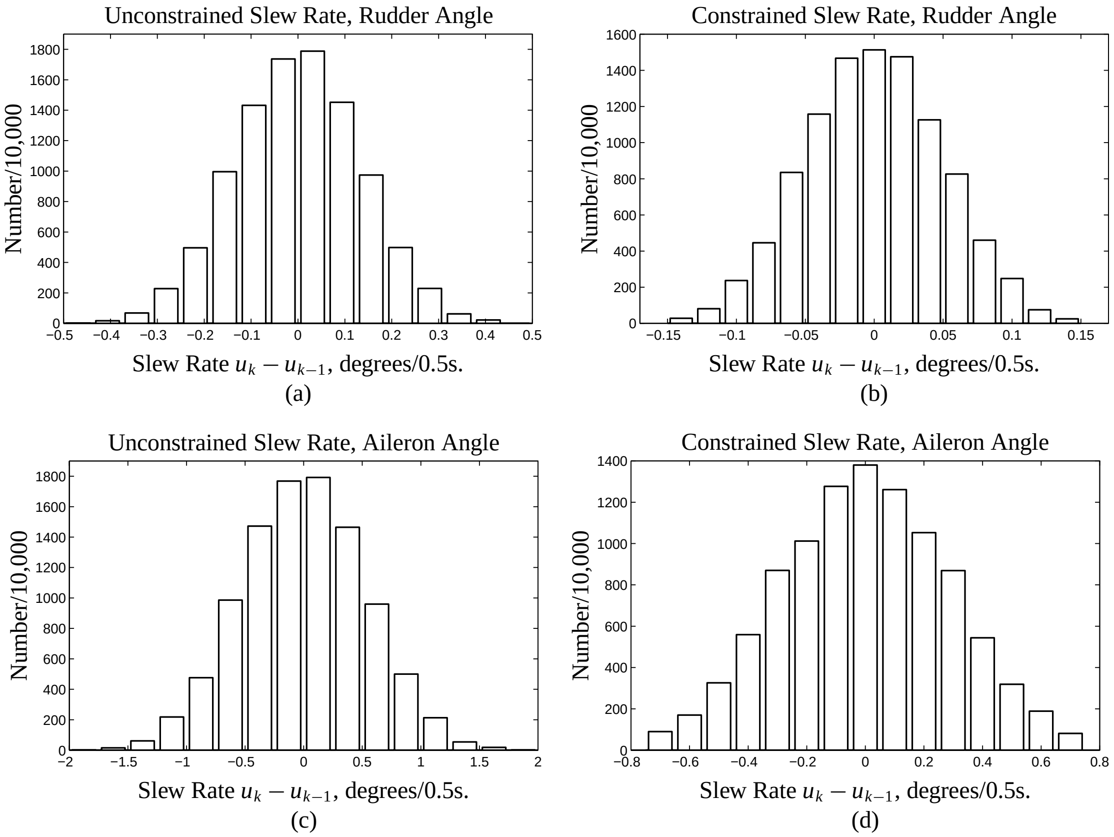


Figure 4.22: Tracking performance and control effort for the Boeing 747 controller. In (a) and (b), the tracking performance is shown. The desired response is a gray line, the actual discrete-time response is plotted as dots at each sample instant, and the continuous-time response is plotted as a solid line. In (c) and (d), the corresponding control effort is plotted. The controller had no constraints on the control effort.

minimum-phase tank example, there is some ringing in the continuous-time response so control-effort constraints may be desired to remove the ringing. Constraints on the control effort might also be considered for all of the other reasons discussed earlier.

A slew-rate constraint was placed on the control effort. The constraints are limits on the control effort such that $-0.15 \leq$ (slew-rate of rudder angle) ≤ 0.15, and $-0.75 \leq$ (slew-rate of aileron angle) ≤ 0.75. The constraints were implemented using the following penalty functions:

$$h\left(u_k^{(\text{rudder})}\right) = \begin{cases} \left(\frac{\left(u_k^{(\text{rudder})} - u_{k-1}^{(\text{rudder})}\right) + 0.1485}{-0.15 + 0.1485}\right)^2, & \text{if } u_k^{(\text{rudder})} < -0.1485; \\ \left(\frac{\left(u_k^{(\text{rudder})} - u_{k-1}^{(\text{rudder})}\right) - 0.1485}{0.15 - 0.1485}\right)^2, & \text{if } u_k^{(\text{rudder})} > 0.1485; \\ 0, & \text{otherwise,} \end{cases}$$



(a) (b) (c) (d)

Figure 4.23: Histograms of constrained and unconstrained Boeing 747 slew rate.

$$h\big(u_k^{\text{(aileron)}}\big) = \begin{cases} \left(\dfrac{\big(u_k^{\text{(aileron)}} - u_{k-1}^{\text{(aileron)}}\big) + 0.7425}{-0.75 + 0.7425}\right)^2, & \text{if } u_k^{\text{(aileron)}} < -0.7425; \\ \left(\dfrac{\big(u_k^{\text{(aileron)}} - u_{k-1}^{\text{(aileron)}}\big) - 0.7425}{0.75 - 0.7425}\right)^2, & \text{if } u_k^{\text{(aileron)}} > 0.7425; \\ 0, & \text{otherwise.} \end{cases}$$

Figure 4.23 shows histograms of slew rate for unconstrained and constrained control. As can be seen, the constraints are satisfied by the converged controller.

4.4.4 Nonlinear SISO Plant

A fourth set of simulation experiments was performed to demonstrate control of a SISO nonlinear system. The goal was to control the heading angle of a large oceangoing ship (Sec. 3.3.1), with

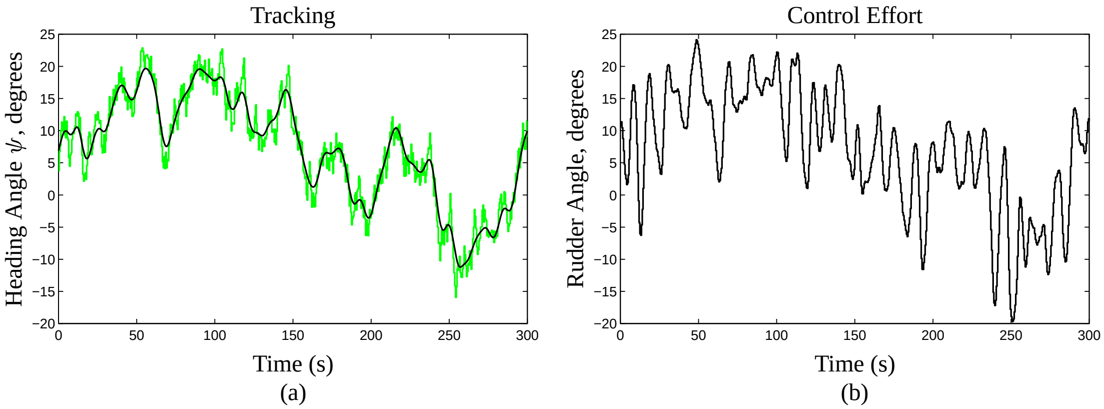


Figure 4.24: Tracking performance of ship controller. The reference model is a delay of 12 samples (6 seconds).

constraints on the maximum rudder angle and the rate-of-change of the rudder angle. This example is unique among all the examples chosen in that the constraints were built into the dynamics of the ship, and thus no external penalty function was used to adapt the controller to perform constrained control. This method worked very well, and guaranteed that the constraints would be met, regardless of the control input signal. It is not recommended for linear plants because it implicitly causes the plant dynamics to become nonlinear, and thus a linear plant model is no longer feasible and a linear controller will no longer work well.

The ship was commanded to track a first-order Markov process which generated the desired heading angle. The Markov process had a pole at $z = 0.99$ and was fed by i.i.d. uniform random numbers with maximum magnitude 0.05. A $\mathcal{N}_{(22,1):50:1}$ controller was trained to control the ship, with a command latency of 12 time samples. Figure 4.24(a) shows the tracking performance for a typical input signal. The gray line is the desired heading angle, and the solid line is the actual heading angle as a function of time. We see that the built-in constraints on the control effort do not allow the ship to follow all of the high-frequency peaks in the desired response signal, but tracking of the moving average is very good.

This plant is nonminimum-phase. The meaning of "nonminimum-phase" in the context of nonlinear control is that a stable, causal inverse does not exist. We must use a delay in the reference model in order to provide good control. Figure 4.25 plots steady-state mean-squared system error versus system latency. Different nonlinear controllers, with architectures $\mathcal{N}_{(10+\Delta,1):50:1}$, were trained for desired response delays of five through sixteen. The steady-state mean-squared system error was measured for each controller, and the results plotted. We can see that the error drops until the delay is about 14 time samples (7 seconds). After that point, the error remains constant. This

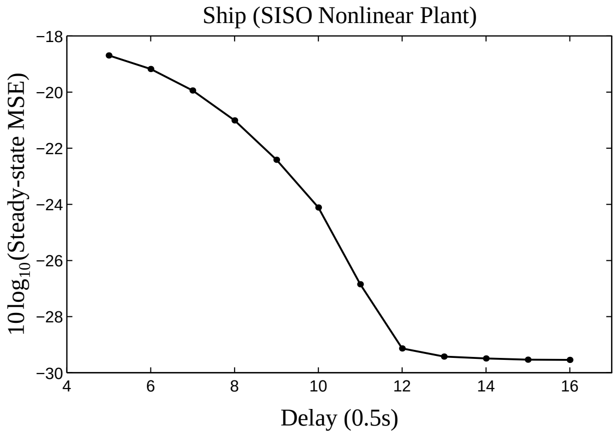


Figure 4.25: Steady-state system mean-squared error versus the system delay. The steady-state mean-squared error decreases to a constant value.

result is in accord with the analytical result proven for constrained linear controllers—the mean-squared system error cannot improve past a certain point because the constraints on the control effort prohibit it. In this example, a delay of 12 time samples (6 seconds) seemed a good compromise between latency and system error, so was used in the simulation of Fig. 4.24.

4.4.5 Nonlinear MIMO Plant

The final example is for constrained control of a nonlinear MIMO plant. This plant is the two-link robotic arm of Sec. 3.3.2. The arm is commanded to track a given time trajectory of desired arm angles. A higher-level controller is assumed to compute the inverse kinematics to provide the desired arm angles. The adaptive inverse controller computes control signals to make the arm dynamically track this desired trajectory.

Both joints were commanded to follow desired trajectories which were independent first-order Markov processes. Both processes were generated with a pole at $z = 0.95$ and fed with i.i.d. uniform random variables with zero mean and maximum magnitude 27°. Robots are required to perform very high-speed accurate control, so are minimum-phase systems by design. Thus, the reference model was a unit delay.

A $\mathcal{N}_{(11,1):20:2}$ controller was trained to control the arm. The tracking response and control effort for a characteristic input signal are plotted in Fig. 4.26. In (a) and (b), the desired response is a gray line, the actual response is a solid line. Very precise control is achieved, but at the expense of high control effort, and in particular, high slew rate.

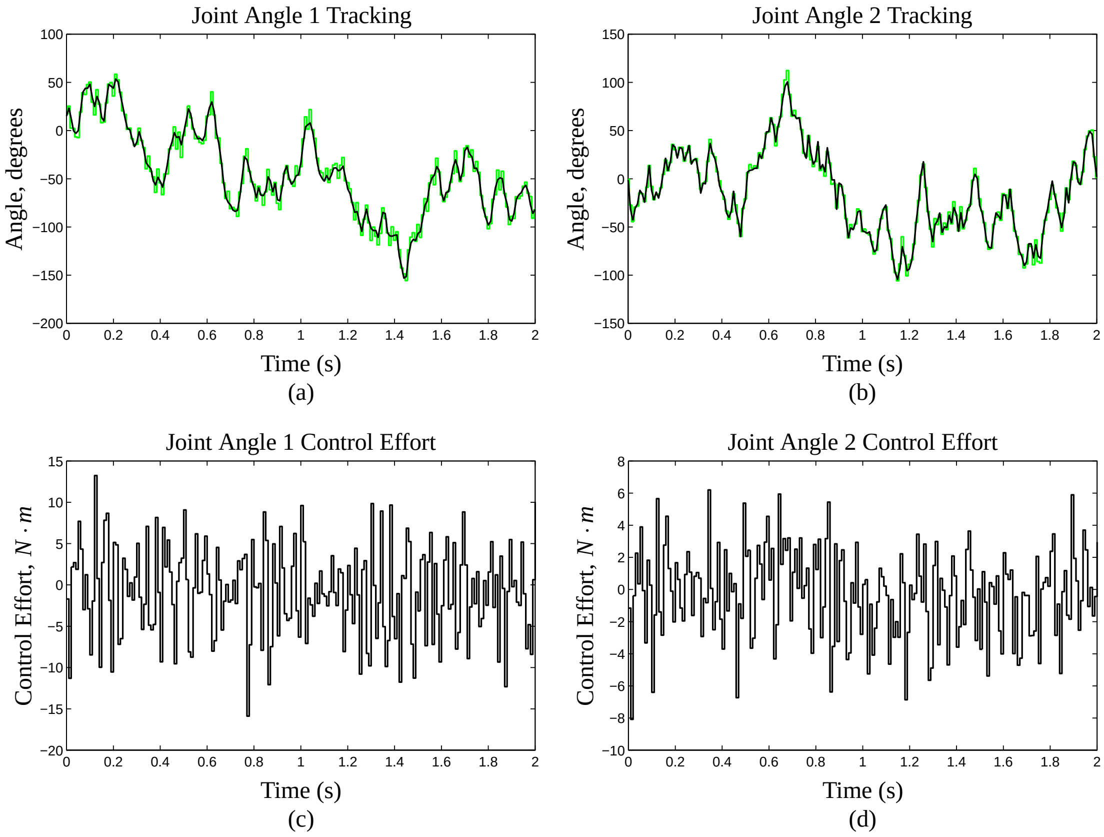
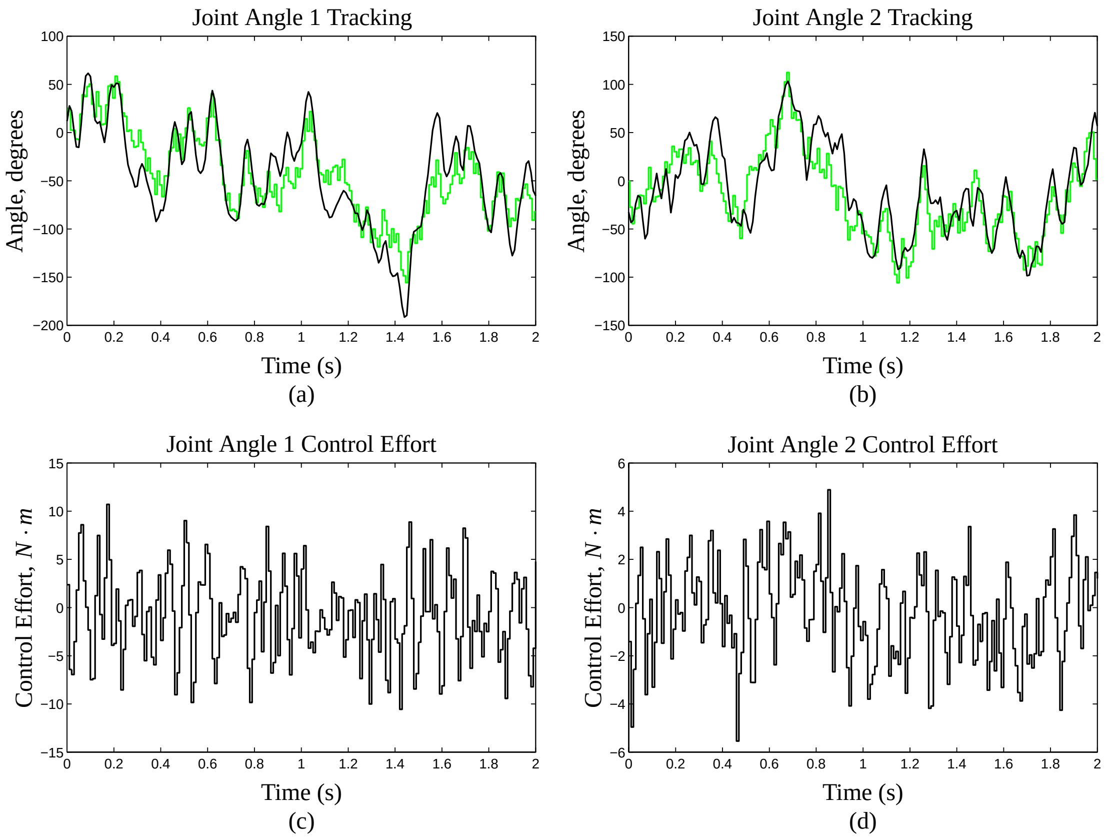


Figure 4.26: Tracking performance and control effort for the robot controller. In (a) and (b), the tracking performance is shown. The desired response is a gray line, the actual response is plotted as a solid line. In (c) and (d), the corresponding control effort is plotted. The controller had no constraints on the control effort.

A controller was adapted where the slew rate was limited. The goal was to limit the slew rate of the first angle to $\pm 15 N \cdot m / 10ms$, and to limit the slew rate of the second angle to $\pm 5 N \cdot m / 10ms$. The penalty function is similar to the ones used in previous examples:

$$h\left(u_k^{(1)}\right) = \begin{cases} \left(\frac{\left(u_k^{(1)} - u_{k-1}^{(1)}\right) + 14.85}{-15 + 14.85}\right)^2, & \text{if } u_k^{(1)} < -14.85; \\ \left(\frac{\left(u_k^{(1)} - u_{k-1}^{(1)}\right) - 14.85}{15 - 14.85}\right)^2, & \text{if } u_k^{(1)} > 14.85; \\ 0, & \text{otherwise,} \end{cases}$$



(a) (b) (c) (d)

Figure 4.27: Tracking performance and control effort for the constrained robot controller. In (a) and (b), the tracking performance is shown. The desired response is a gray line, the actual response is plotted as a solid line. In (c) and (d), the corresponding control effort is plotted. The controller had slew-rate constraints on the control effort.

$$h\left(u_k^{(2)}\right) = \begin{cases} \left(\frac{\left(u_k^{(2)} - u_{k-1}^{(2)}\right) + 4.95}{-5+4.95}\right)^2, & \text{if } u_k^{(2)} < -4.95; \\ \left(\frac{\left(u_k^{(2)} - u_{k-1}^{(2)}\right) - 4.95}{5-4.95}\right)^2, & \text{if } u_k^{(2)} > 4.95; \\ 0, & \text{otherwise.} \end{cases}$$

The tracking response of the constrained controller is plotted in Fig. 4.27. We see that the response is not as accurate as the unconstrained response of Fig. 4.26, but still tracks reasonably well. Histograms of slew-rate for the control signals of the constrained and unconstrained controllers are shown in Fig. 4.28, and we see that the constrained controller satisfies the constraints.

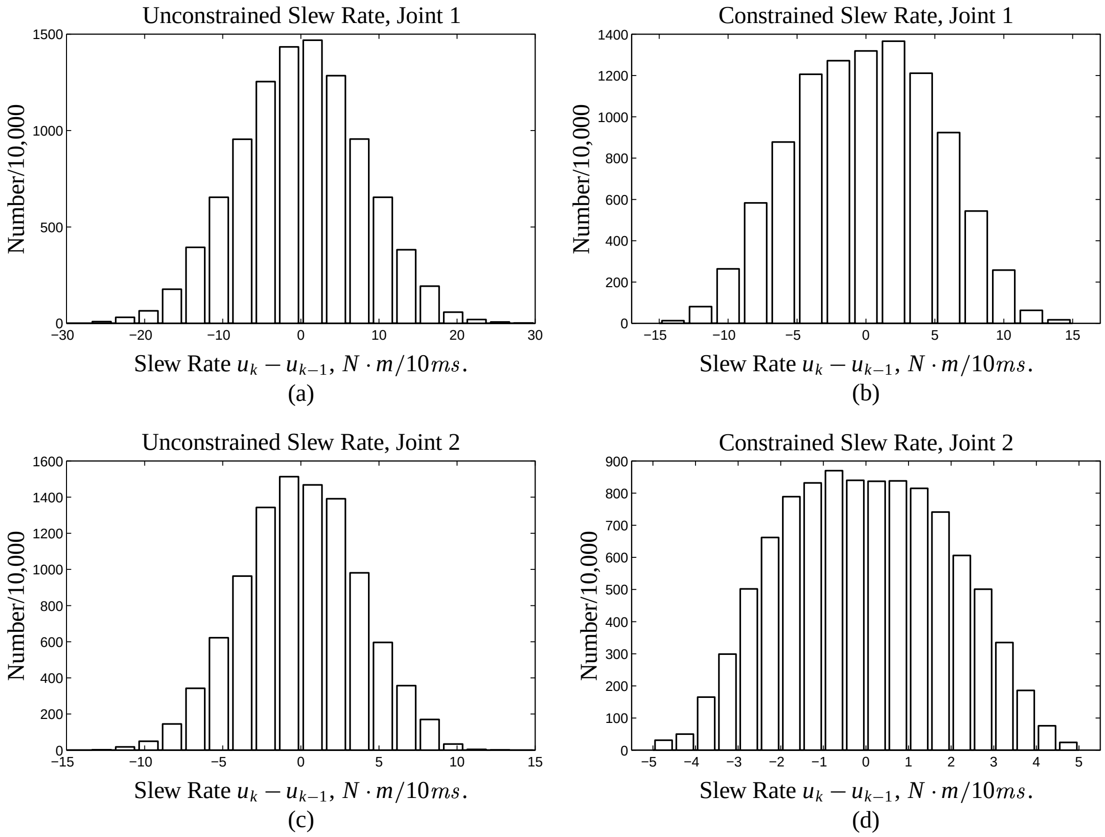


Figure 4.28: Histograms of slew rate using constrained and unconstrained controllers for the robot example.

4.5 Summary

This chapter has three main divisions. The first is an analytic discussion of constrained control; the second derives an algorithm to perform constrained control in the adaptive inverse control paradigm; and the third demonstrates this algorithm with a wide variety of examples.

It was shown analytically, and verified with simulations, that precision of control comes at the cost of high control effort. If very precise control is desired, the actuator signals are very large. Problems with large control effort include (1) The actuator may not be able to respond to the control command due to its physical design, thus causing degradation in the control which is not accounted for in the design; and (2) The actuator or the system being controlled may be damaged by excessive control effort. Since this is a significant problem, a method is devised to perform adaptive inverse control with constraints on the control effort. The controller is adapted such that the mean-squared system error is minimized under the constraint that the peak control effort and/or the peak slew rate be lower than a user-specified amount. Simulations have shown that this works very well.

If the plant is nonminimum-phase, its inverse does not exist. However, if a delay in the control action is acceptable, then a “delayed inverse” does exist, and very precise control can be performed. Choosing the correct delay is a significant design issue. This chapter gives a very simple method of choosing this latency if there are constraints on the control effort. Analytical results and simulations agree that there is a value of latency beyond which control precision does not improve. We choose this delay as the optimal compromise between precision of control and control latency. Finally, Table. 4.1 tabulates the architectures of the plant models and controllers used in simulations in this chapter. The overall conclusion is that very good feedforward control may be achieved.

TABLE 4.1 FILTER ARCHITECTURES USED IN THIS CHAPTER.

System	Plant Model, $\widehat{P}$	Linear Feedforward Controller, C	Nonlinear Feedforward Controller, C
Linear SISO			
Unconstrained Minimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(20,0):1}$	—
Constrained Minimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(20,0):1}$	$\mathcal{N}_{(4,1):20:1}$
Unconstrained Nonminimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(15,0):1}$	—
Constrained Nonminimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(15,0):1}$	$\mathcal{N}_{(20,1):20:1}$
Linear MIMO			
Unconstrained	$\mathrm{FIR}_{(60,0):2}$	$\mathrm{FIR}_{(128,0):2}$	—
Constrained	$\mathrm{FIR}_{(60,0):2}$	$\mathrm{FIR}_{(128,0):2}$	—
Nonlinear SISO			
Constrained Dynamics	$\mathcal{N}_{(20,1):8:1}$	—	$\mathcal{N}_{(10+\Delta,1):50:1}$
Nonlinear MIMO			
Unconstrained	$\mathcal{N}_{(20,0):10:2}$	—	$\mathcal{N}_{(11,1):20:2}$
Constrained	$\mathcal{N}_{(20,0):10:2}$	—	$\mathcal{N}_{(11,1):20:2}$

Chapter 5

Closing the Loop: Disturbance Cancelling

Sometimes the news is in the noise, and sometimes the news is in the silence.
—Thomas L. Friedman in *New York Times*

5.1 Introduction

It has now been established that a disturbance-free dynamical system may be controlled with a feedforward adaptive controller. The plant output tracks the desired output as closely as possible in a mean-squared-error sense. It remains to determine what can be done to mitigate plant disturbance should it be present. One basic method has been used in the past for disturbance cancelling with linear plants [45]. This method is simplified and enhanced in the following pages. Three methods have been attempted in the past to cancel disturbances with nonlinear plants. The first [45], based on a derivative plant model, suffers from high complexity; the second [45], based on the filtered-epsilon method, has been demonstrated to fail [4]; and the third [2], based on *internal model control* [9, 10, 11, 34, 5, 6] is shown in Sec. 5.2.2 to be incorrect if on-line plant modeling is performed. These three approaches are abandoned here in favor of extending the disturbance cancelling method used for linear plants to encompass nonlinear plants as well.

When considering how to cancel disturbances, the first idea which may come to mind is to simply "close the loop." Two methods commonly used to do this are discussed in Sec. 5.2.2. Unfortunately, neither of these methods is appropriate if an adaptive controller is being designed. Closing the loop will cause the controller to adapt to a "biased" solution. It is shown here that the extent of the bias is dependent on the plant dynamics, the spectrum of the disturbance, and the spectrum of the plant's control signal. Since these combine in a fairly complicated way, simulation examples are presented as an attempt to show the significance of the problem.

There is no need to fear! A method is developed which improves upon previous "optimal" results for disturbance cancelling with linear plants. It is highly effective for nonlinear systems as well. In this chapter, some analysis is first performed to demonstrate that conventional disturbance rejection methods fail when simultaneous on-line system identification and disturbance rejection is performed. Analysis is done on an alternate technique, which is optimal for linear systems, but slightly sub-optimal for nonlinear systems. Methods for adapting the disturbance canceller are introduced, and simulations are presented to verify the analysis and the synthesis methods.

5.2 Analysis of Disturbance Cancelling

5.2.1 Correctness of Feedforward Design in the Presence of Disturbance

Until this point, disturbance has been ignored. We have not done anything to remove it and we have not even checked to see if the design presented thus far "works" if there is disturbance. In this section we present a proof to show that the controller adapted according to Chap. 4 causes the plant output to converge to the desired output plus the disturbance. That is, the design is correct but does nothing to mitigate disturbance.

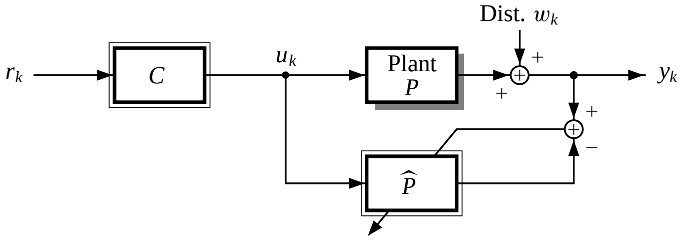


Figure 5.1: Plant modeling in the context of feedforward control. The circuitry for adapting C has been omitted for clarity.

We re-examine the system identification problem as depicted in Fig. 5.1. To be most general in our analysis, we assume that the plant is nonlinear MIMO and that the adaptive controller and adaptive plant model are also nonlinear MIMO systems. We first confirm that the adaptive plant model $\widehat{P}$ converges to P. From Sec. 2.3.3 we know that the optimal solution for $\widehat{P}$ is:

$$\begin{aligned}\widehat{P}^{(\mathrm{opt})}(\vec{u}_k) &= \mathbb{E}\big[y_k \mid \vec{u}_k\big]\\ &= \mathbb{E}\big[P(\vec{u}_k)+w_k \mid \vec{u}_k\big]\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}\big[P(\vec{u}_k) \bigm| \vec{u}_k\big] + \mathbb{E}\big[w_k \bigm| \vec{u}_k\big] \\
&= P(\vec{u}_k) + \mathbb{E}\big[w_k\big] \\
&= P(\vec{u}_k),
\end{aligned}
$$

where $\vec{u}_k$ is an infinite vector containing past values of the control signal $u_k, u_{k-1}, \ldots$, and where two assumptions were made: (1) In the fourth line we assume that the disturbance is statistically independent of the command input signal u_k; and (2) In the final line we assume that the disturbance is zero-mean.[1] Under these two assumptions, the plant model converges to the plant despite the presence of disturbance. Similarly, it can be shown (using a method similar to the one in Sec. 5.2.2) that the controller adapts to the correct solution, unbiased by the disturbance. The conclusion is that the plant output converges to the sum of the desired output *plus* the disturbance. We now investigate how to reduce disturbance.

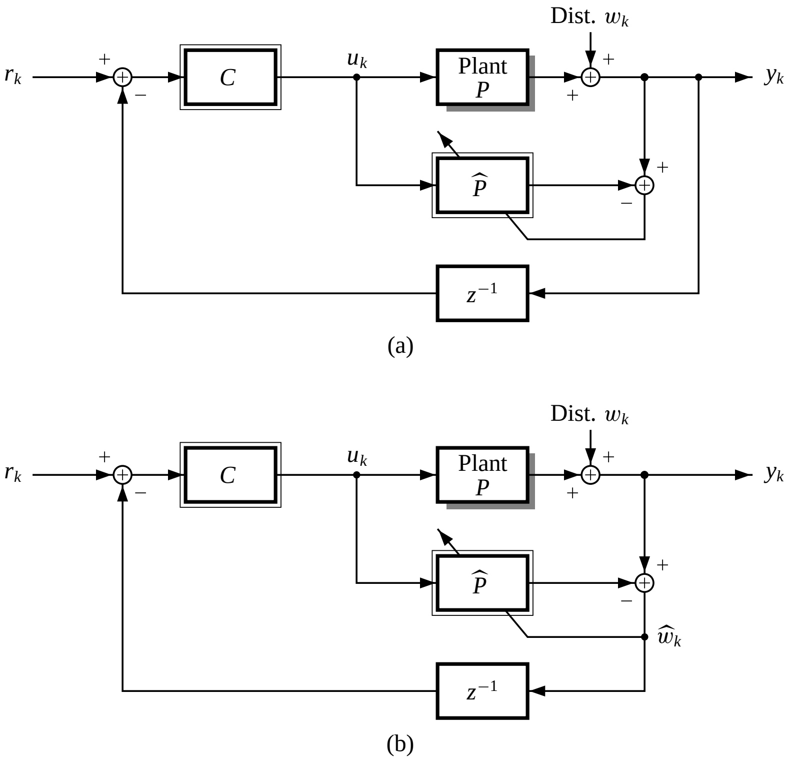
5.2.2 Conventional Disturbance Rejection Methods Fail

The block diagram for a feedforward control system is shown in Fig. 5.1. To complete the control design, and reject the disturbance, the conventional approach is to "close the loop." Two approaches commonly seen in the literature are shown in Fig. 5.2.[2] By closing the loop we either feed back the disturbed plant output y_k, as in Fig. 5.2(a), or we feed back an estimate of the disturbance $\widehat{w}_k$, as in Fig. 5.2(b). The approach shown in Fig. 5.2(a) is more conventional, but is difficult to use with adaptive inverse control since the transfer function of the closed-loop system is dramatically different from the transfer function of the open-loop system. Different methods than those presented in this dissertation are required to adapt C. The approach shown in Fig. 5.2(b) is called *internal model control* [9, 10, 11, 34, 5, 6]. The benefit of using this scheme is that the transfer function of the closed-loop system is equal to the transfer function of the open-loop system if the plant model is identical to the plant. Therefore, the methods found to adapt the controller for feedforward control may be used directly.

Unfortunately, closing the loop using *either* method in Fig. 5.2 will cause $\widehat{P}$ to adapt to an incorrect solution. In the following analysis the case of a linear SISO plant controlled with internal model control is considered. A similar analysis may be performed for the conventional feedback system in Fig. 5.2(a), with the same conclusion.

[1] Alternately, we could make the single assumption that the disturbance is conditionally zero-mean given the command input signal; however, this does not make much physical sense.

[2] The purpose of the z^{-1} block will be explained in Sec. 5.2.4.



(a)

(b)

Figure 5.2: Two methods to close the loop: (a) The output, y_k, is fed back to the controller; (b) An estimate of the disturbance, $\widehat{w}_k$, is fed back to the controller. The circuitry for adapting C has been omitted for clarity.

Shannon-Bode Solution for $\widehat{P}$: When the loop is closed as in Fig. 5.2(b), the estimated disturbance term $\widehat{w}_k$ is subtracted from the reference input r_k. The resulting composite signal is filtered by the controller C and becomes the plant input signal, u_k. In the analysis done so far, we have assumed that u_k is independent of w_k, but that assumption is no longer valid. We need to revisit the analysis performed for system identification to see if the plant model $\widehat{P}$ still converges to P.

The direct approach to the problem is to calculate the least mean-squared-error solution for $\widehat{P}$ and see if it is equal to P. However, due to the feedback loop involved, it is not possible to obtain a closed form solution for $\widehat{P}$. An indirect approach is taken here. We do not need to know exactly what $\widehat{P}$ converges to—we only need to know whether or not it converges to P. The following indirect procedure is used a number of times in this chapter:

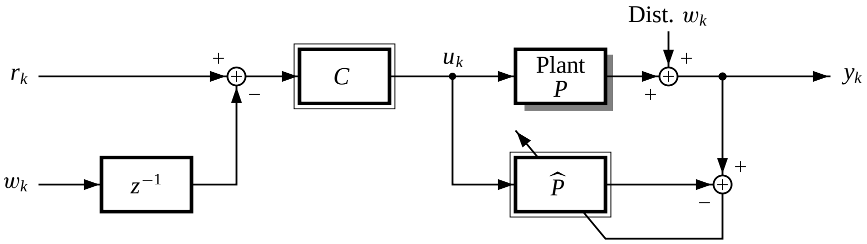


Figure 5.3: An intermediate step when analyzing the convergence of $\widehat{P}$.

1. First, remove the feedback path (open the loop) and perform on-line plant modeling and controller adaptation as shown in Fig. 5.1. When convergence is reached, we know from Sec. 5.2.1 that $\widehat{P} \to P$.

2. At this time, $\widehat{P} \approx P$, and the disturbance estimate $\widehat{w}_k$ is very good. We assume that $\widehat{w}_k = w_k$. This assumption allows us to construct a feedforward-only system which is equivalent to the internal model control feedback system by substituting w_k for $\widehat{w}_k$. This is drawn in Fig. 5.3.

3. Lastly, analyze the system, substituting w_k for $\widehat{w}_k$. If the least mean-squared-error solution for $\widehat{P}$ still converges to P, then the assumption made in the second step remains valid, and the plant is being modeled correctly. If $\widehat{P}$ diverges from P with the assumption made in step 2, then it can not converge to P even if the assumption is not made and an exact analysis is done. We conclude that the assumption is justified for the purpose of checking for proper convergence of $\widehat{P}$.

We now apply this procedure to analyze the system of Fig. 5.2(b). We first open the loop and allow the plant model to converge to the plant. Secondly, we assume that $\widehat{w}_k \approx w_k$. Finally, we compute the least mean-squared-error solution for $\widehat{P}$. If causality is enforced, this solution is the Shannon-Bode solution from Sec. 2.2.3. To calculate it, we first determine the correlation terms $\Phi_{uy}(z)$ and $\Phi_{uu}(z)$.

$$\begin{aligned}
(\phi_{uy})_n &= \mathbb{E}\big[u_k y_{k+n}\big] \\
&= \mathbb{E}\big[u_k \cdot (p_{k+n} * u_{k+n} + w_{k+n})\big] \\
&= p_n * (\phi_{uu})_n + \mathbb{E}\big[u_k w_{k+n}\big],
\end{aligned}$$

where p_k is the impulse response of the plant. To proceed, note that $u_k = c_k * (r_k - w_{k-1}) = c_k * (r_k - w_k * \delta_{k+1})$, where δ_k is the unit impulse function. So,

$$(\phi_{uy})_n = p_n * (\phi_{uu})_n + \mathbb{E}\big[c_k * (r_k - w_k * \delta_{k+1}) \cdot w_{k+n}\big]$$

$$\begin{aligned}
&= p_n * (\phi_{uu})_n - c_{-n} * \delta_{-n-1} * (\phi_{ww})_n \\
\Phi_{uy}(z) &= P(z)\Phi_{uu}(z) - z^{+1}C(z^{-1})\Phi_{ww}(z).
\end{aligned}$$

Similarly, we compute $\Phi_{uu}(z)$.

$$\begin{aligned}
(\phi_{uu})_n &= \mathbb{E}\big[u_k u_{k+n}\big] \\
&= \mathbb{E}\big[c_k * (r_k - w_{k-1}) \cdot c_{k+n} * (r_{k+n} - w_{k+n-1})\big] \\
&= c_{-n} * c_n * \big((\phi_{rr})_n + (\phi_{ww})_n\big) \\
\Phi_{uu}(z) &= C(z)C(z^{-1})\big(\Phi_{rr}(z) + \Phi_{ww}(z)\big).
\end{aligned}$$

The Shannon-Bode solution for $\widehat{P}$ may now be calculated (assuming in the third line that the plant is causal)

$$\begin{aligned}
\widehat{P}^{(\mathrm{opt})}_{\mathrm{causal}}(z) &= \frac{1}{\Phi^{+}_{uu}(z)}\left[\frac{\Phi_{uy}(z)}{\Phi^{-}_{uu}(z)}\right]_{+} \\
&= \frac{1}{\Phi^{+}_{uu}(z)}\left[\frac{P(z)\Phi_{uu}(z) - z^{+1}C(z^{-1})\Phi_{ww}(z)}{\Phi^{-}_{uu}(z)}\right]_{+} \\
&= P(z) - \frac{1}{\Phi^{+}_{uu}(z)}\left[\frac{z^{+1}C(z^{-1})\Phi_{ww}(z)}{\Phi^{-}_{uu}(z)}\right]_{+} \\
&= P(z) - \Delta_{\widehat{P}}(z).
\end{aligned}$$

Assumptions concerning the nature of the r_k and w_k signals are needed to simplify this further. For example, if r_k and w_k are both white, then the plant model converges to the plant. Under almost all other conditions, however, the plant model converges to something else. We conclude that in general, an adaptive plant model made using the internal model control scheme will be biased by disturbance. We next show that the entire control system will then be biased.

Wiener Solution for C: The method used to train C backpropagates the system error through the plant model $\widehat{P}$ in order to create an error signal to adapt C. We can therefore consider the process of adapting C as that of adapting the cascade $(\widehat{P}C)$ to match M, where $\widehat{P}$ is a fixed filter, unchanged by the adaptation of C.

The input to C is: $r'_k = r_k - w_k * \delta_{k+1}$. The desired response of the system is equal to: $m_k * (r_k - w_k * \delta_{k+1})$. We find that the Wiener solution is

$$\begin{aligned}
(\widehat{P}C)^{(\mathrm{opt})}(z) &= \frac{\Phi_{r'd}(z)}{\Phi_{r'r'}(z)} \\
&= \frac{M(z)\big[\Phi_{rr}(z) + z^{-1}\Phi_{ww}(z)\big]}{\big[\Phi_{rr}(z) + z^{-1}\Phi_{ww}(z)\big]} \\
&= M(z).
\end{aligned}$$

Therefore, C is adapted such that $(\widehat{P}C)(z) = M(z)$. This means that $C^{(\text{opt})}(z) = M(z)/\widehat{P}(z)$, or

$$C^{(\text{opt})}(z) = \frac{M(z)}{P(z) - \Delta_{\widehat{P}}(z)} \neq \frac{M(z)}{P(z)}.$$

The controller and the entire system dynamics are biased by the disturbance.

The bottom line: If the loop is closed, and the plant model is allowed to continue adapting after the loop is closed, the overall control system will become biased by the disturbance. One solution is to "freeze" the weights of the plant model just before the loop is closed. This solution will work, but does not allow the control system to respond to time variations in the plant dynamics. Another solution, applicable to linear plants, is to perform plant modeling with dither signals rather than with the command input signal u_k [45, schemes B,C]. However, this will increase the output noise level. A better solution is presented next, in which the plant model is allowed to continue adaptation. There is no extra disturbance at the output. The plant disturbances will be handled by a separate circuit from the one handling the task of dynamic response. This results in an overall control problem which is partitioned in a very nice way.

5.2.3 A Solution Allowing On-Line Adaptation of $\widehat{P}$

The only means at our disposal to cancel disturbance is through the plant input signal u_k. This signal must be computed in such a way that the plant output negates (as much as possible) the disturbance. Therefore, the plant input signal must be statistically dependent on the disturbance. However, we have just seen that the input signal to the plant model $\widehat{P}$ cannot contain terms correlated with the disturbance or the plant model will be biased. This conundrum was first solved by Widrow [45] and his solution is shown (slightly modified) in Fig. 5.4.

By studying the figure, we see that the control scheme is very similar to internal model control. The main difference is that the feedback loop is "moved" in such a way that the disturbance dynamics do not appear at the input to the adaptive plant model, but do appear at the input to the plant. That is, the controller output u_k is used as input to the adaptive plant model $\widehat{P}$; on the other hand, the input to the plant is equal to $u_k + \tilde{u}_k$, where $\tilde{u}_k$ is the output of a special disturbance-cancelling circuit, X. $\widehat{P}$ is not used directly to estimate the disturbance; rather, a filter whose weights are a digital copy of those in $\widehat{P}$ is used to estimate the disturbance. This filter is called $\widehat{P}_{\text{COPY}}$.

In later sections we will see how to adapt X and how to modify the diagram if the plant is nonlinear. Now, we proceed to show that the design is correct if the plant is linear.

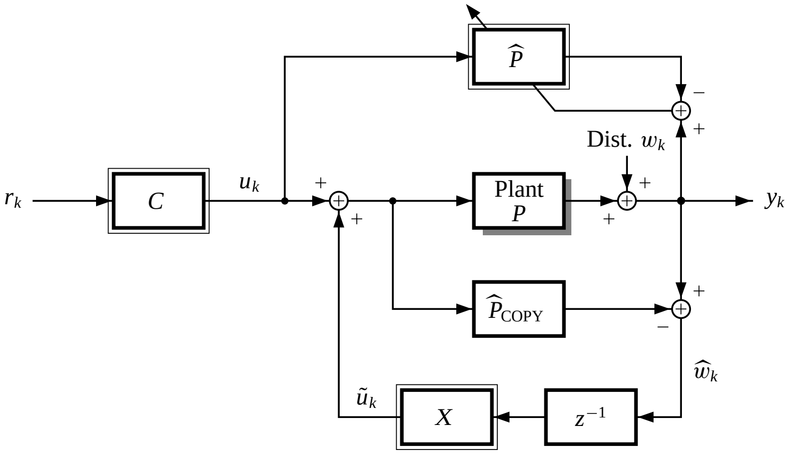


Figure 5.4: Correct on-line adaptive plant modeling in conjunction with disturbance cancelling for linear plants. The circuitry for adapting C has been omitted for clarity.

Linear Plant Model Unbiased by Disturbance

Here, we show that the scheme of Fig. 5.4, when used with a linear plant, causes the plant model to adapt to the correct solution. As before, we follow the three-step procedure. First, we open the loop and allow $\widehat{P}$ to adapt until it converges to P. Secondly, we assume that $\widehat{w}_k \approx w_k$. Lastly, we check to see whether or not $\widehat{P}$ remains converged to P.

The input to the plant model is u_k. The signal used as the desired output when adapting $\widehat{P}$ is the disturbed plant output

$$y_k = (u_k + \tilde{u}_k) * p_k + w_k,$$

where p_k is the impulse response of the plant, and $\tilde{u}_k$ is the output of the disturbance cancelling filter X. This latter term is computed to be

$$\tilde{u}_k = x_k * w_k * \delta_{k+1},$$

where x_k is the impulse response of the disturbance cancelling filter X, and δ_{k+1} is a unit impulse at time $k = -1$, representing a unit delay. We compute the Shannon-Bode solution for the adaptive plant model by first computing $\Phi_{uy}(z)$.

$$\begin{aligned}
(\phi_{uy})_n &= \mathbb{E}\big[u_k\,(u_{k+n} * p_{k+n} + \tilde{u}_{k+n} * p_{k+n} + w_{k+n})\big] \\
&= p_n * (\phi_{uu})_n \\
\Phi_{uy}(z) &= P(z)\Phi_{uu}(z),
\end{aligned}$$

assuming that u_k and w_k are uncorrelated. The Shannon-Bode solution is

$$\begin{aligned}\widehat{P}^{(\text{opt})}_{\text{causal}}(z) &= \Big[\big(P(z)\Phi_{uu}(z)\big)\big(\Phi^{-}_{uu}(z)\big)^{-1}\Big]_{+}\big(\Phi^{+}_{uu}(z)\big)^{-1}\\ &= P(z),\end{aligned}$$

assuming that the plant is causal. So, the plant model converges to the plant regardless of the disturbance (if it is zero-mean and uncorrelated with u_k) and regardless of X.

Nonlinear Plant Model Still Biased

We now examine the least mean-squared-error solution for the plant model if the plant is a nonlinear dynamical system. Again, we use the three-step procedure. We first open the loop and allow $\widehat{P}$ to adapt until it converges to P. Secondly, we substitute $\widehat{w}_k \approx w_k$. Finally we check to see if $\widehat{P}$ still converges to P, using the techniques from Sec. 2.3.3. As before, we assume that w_k is independent of u_k, and is zero-mean.

$$\begin{aligned}\widehat{P}^{(\text{opt})}(\vec{u}_k) &= \mathbb{E}\big[y_k \mid \vec{u}_k\big]\\ &= \mathbb{E}\big[P(\vec{u}_k+\vec{\tilde{u}}_k)+w_k \mid \vec{u}_k\big]\\ &= \mathbb{E}\big[P(\vec{u}_k+\vec{\tilde{u}}_k) \mid \vec{u}_k\big]+\mathbb{E}\big[w_k \mid \vec{u}_k\big]\\ &= \mathbb{E}\big[P(\vec{u}_k+\vec{\tilde{u}}_k) \mid \vec{u}_k\big],\end{aligned}$$

where $\vec{u}_k$ is an infinite vector containing past values of the control signal $u_k, u_{k-1}, \ldots$, and $\vec{\tilde{u}}_k$ is an infinite vector containing past values of the disturbance canceller output, $\tilde{u}_k, \tilde{u}_{k-1}, \ldots$ Using a Taylor series expansion of the system dynamics at each time step around the "point" $\vec{u}_k$ we get

$$\begin{aligned}\widehat{P}^{(\text{opt})}(\vec{u}_k) &= \mathbb{E}\Big[P(\vec{u}_k)+\vec{\tilde{u}}_k^T P'(\vec{u}_k)+0.5\vec{\tilde{u}}_k^T P''_k(\vec{u}_k)\vec{\tilde{u}}_k+\ldots \,\Big|\, \vec{u}_k\Big]\\ &= P(\vec{u}_k)+\mathbb{E}\Big[\vec{\tilde{u}}_k^T P'(\vec{u}_k)+0.5\vec{\tilde{u}}_k^T P''_k(\vec{u}_k)\vec{\tilde{u}}_k+\ldots \,\Big|\, \vec{u}_k\Big].\end{aligned}$$

Since w_k is assumed to be independent of u_k, then $\vec{\tilde{u}}_k$ is independent of $\vec{u}_k$. We also assume that $\vec{\tilde{u}}_k$ is zero-mean.

$$\begin{aligned}\widehat{P}^{(\text{opt})}(\vec{u}_k) &= P(\vec{u}_k)+\underbrace{\mathbb{E}\big[\vec{\tilde{u}}_k^T\big]}_{=0}P'(\vec{u}_k)+\mathbb{E}\big[0.5\vec{\tilde{u}}_k^T P''_k\vec{\tilde{u}}_k \mid \vec{u}_k\big]+\text{H.O.T.} \qquad (5.1)\\ &= P(\vec{u}_k)+\text{H.O.T.}\\ &\approx P(\vec{u}_k),\end{aligned}$$

where "H.O.T." means "higher-order terms." The first order term is zero. The higher-order terms go to zero since, by assumption, the plant is stable and its Taylor series expansion exists and is finite. The higher-order terms may not go to zero quickly unless the plant is approximately linear, in which case they disappear and the plant model approximates the true plant. If the plant is very nonlinear, however, the plant model may be quite different from the plant.

So, we conclude that this system is not as good as desired if the plant is nonlinear. It may not be possible to perform on-line adaptive plant modeling while performing disturbance cancelling while retaining an unbiased plant model. One solution might involve freezing the weights of the plant model for long periods of time, and scheduling shorter periods for adaptive plant modeling with the disturbance canceller turned off when requirements for output quality are not as stringent. Another solution suggests itself from Eq. (5.1). If the $\tilde{u}_k$ terms are kept small, the bias will disappear. We will see in Sec. 5.3 that X may be adapted using the BPTM algorithm. We can enforce constraints on the output of X using our knowledge from Chap. 4 and ensure that $\tilde{u}_k$ remains small.

5.2.4 Structure of the Disturbance Canceller

This final section of analysis concerns itself with the mathematical function that the disturbance cancelling circuit must compute. A little careful thought in this direction leads to a great deal of insight, and some surprising conclusions are reached. First, we must consider some issues of timing which arise since we are performing discrete-time control. Then, we are ready to investigate the function of the disturbance canceller.

Issues of Timing

The type of adaptive inverse control examined in this dissertation is implemented by a discrete-time digital controller. Due to the discrete-time nature of the control scheme, a subtle issue arises which is not present in continuous-time control systems. Consider the timing diagram in Fig. 5.5.

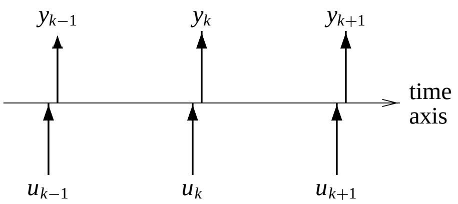


Figure 5.5: Input-output timing of a discrete-time control system.

If the sampling rate of the system is equal to $1/T$ samples per second, then the discrete-time/continuous-time correspondence is: $t = kT$ seconds, where t is the physical time in seconds and k is the discrete-time index. The kth command to the plant u_k takes place at $t = kT$, and the kth plant output y_k is sampled in the neighborhood of $t = kT$ seconds. More precisely, supposing that the plant might not be strictly proper, y_k must be measured at time $t = (kT)^+$ seconds.

The command input to the plant u_k takes finite time to compute, so r_k must be supplied at $t = (kT)^-$ seconds to be able to compute the disturbance-cancelling signal u_k in time. We must also be able to compute $\tilde{u}_k$ slightly before time $t = kT$. This brings us to the important point: *If we want to cancel disturbance, we must have an estimate of w_k $\big($the disturbance actually present at time $t = (kT)^+\big)$ at time $t = (kT)^-$ to be able to compute a command $\tilde{u}_k$ to cancel it.* We do not sample y_k in time to compute w_k from it, so we must use y_{k-1} to estimate the disturbance at time $t = kT$. To make this inherent delay more explicit, we incorporate a z^{-1} block in the feedback path. The delay is unavoidable in any discrete-time control system.

Heuristically Determined Structure of *X*

This section is a heuristic investigation of the function performed by the disturbance canceller X. The analysis is precise if the plant is minimum phase (that is, if it has a stable, causal inverse), but is merely qualitative if the plant is nonminimum phase. The goal of this analysis is not to be quantitative, but rather to develop an understanding of the function performed by X.

A useful way of considering the overall system is drawn in Fig. 5.6. Using operator notation, we restate that the control goal is for X to produce an output so that $y_k = M(\vec{r}_k)$. We can express y_k as

$$y_k = w_k + P\left(C(\vec{r}_k) + X(\vec{\widehat{w}}_{k-1}, \vec{u}_k)\right).$$

Note that the dashed line in the figure shows that X takes the optional signal u_k. This signal is used when controlling nonlinear plants as it allows the disturbance canceller some knowledge of the plant state. We will see shortly that it is not required if the plant is linear.

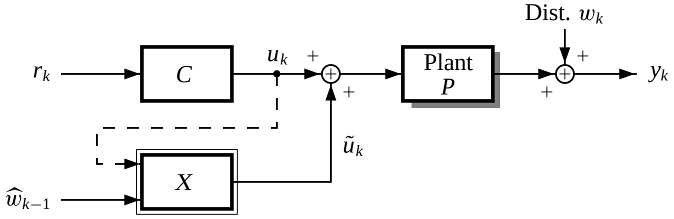


Figure 5.6: A useful way of looking at the feedforward system dynamics.

Next, we substitute $y_k = M(\vec{r}_k)$ and rearrange to solve for the desired response of X. We see that

$$\begin{aligned} X^{(\text{opt})}(\vec{\widehat{w}}_{k-1}, \vec{u}_k) &= P^{-1}\big(M(\vec{r}_k) - \vec{w}_k\big) - C(\vec{r}_k) \\ &= P^{-1}\big(P(\vec{u}_k) - \vec{w}_k\big) - u_k, \end{aligned}$$

assuming that the controller has adapted until $P(\vec{u}_k) \approx M(\vec{r}_k)$. The function of X is a deterministic combination of known (by adaptation) elements P and P^{-1}, but also of the unknown signal w_k. Because of the inherent delay in discrete-time systems, we only know w_{k-1} at any time, so w_k must be estimated from previous samples of $w_{k-1}, w_{k-2}, \ldots$ Assuming that the adaptive plant model is perfect, and that the controller has been adapted to convergence, the internal structure of X is then shown in Fig. 5.7. The w_k signal is computed by estimating its value from previous samples of $\widehat{w}_k$. These are combined and passed through the plant inverse to compute the desired signal $\tilde{u}_k$.

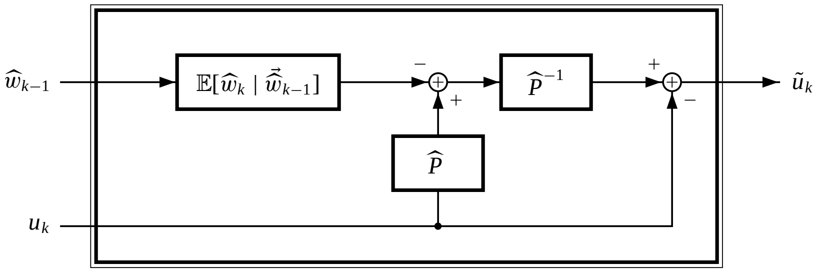


Figure 5.7: Internal structure of X.

Thus, we see that the disturbance canceller contains two parts. The first part is an estimator part which depends on the dynamics of the disturbance source. The second part is the canceller part which depends on the dynamics of the plant. The diagram simplifies for a linear plant since some of the circuitry cancels. Figure 5.8 shows the structure of X for a linear plant.

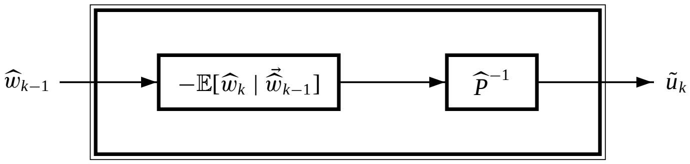


Figure 5.8: Internal structure of X if the plant is linear.

One very important point to notice is that the disturbance canceller still depends on both the disturbance dynamics and the plant dynamics. If the process generating the disturbance is not generated by filtering white noise using a linear filter, then the estimator required will in general be a nonlinear function. The conclusion is that the disturbance canceller should be implemented as a nonlinear filter *even if the plant is a linear dynamical system*. Simulations in Sec. 5.4 demonstrate this result.

If the plant is generalized minimum phase (minimum phase with a constant delay, as with the first example of Sec. 3.2.1), then this solution must be modified slightly. The plant inverse must be a delayed plant inverse, with the delay equal to the delay inherent in the plant. The estimator must estimate the disturbance one time step into the future plus the delay of the plant. For example, if the plant is strictly proper, there will be at least one delay in the plant's impulse response, and the estimator must predict the disturbance at least two time steps into the future.

It was stated earlier that these results are heuristic and do not directly apply if the plant is nonminimum phase. We can see this easily now, since a plant inverse does not exist. However, the results still hold qualitatively since a delayed inverse exists; the solution for X is similar to the one for a generalized minimum phase plant. The structure of X consists of a part depending on the dynamics of the system which amounts to a delayed plant inverse, and a part which depends on the dynamics of the disturbance generating source, which must now predict farther into the future than a single time step. Unlike the case of the generalized minimum phase plant, however, these two parts do not necessarily separate. That is, X implements some combination of predictors and delayed inverses which compute the least mean-squared-error solution.

5.3 Synthesis of the Disturbance Canceller via the BPTM Algorithm

We have seen how a disturbance cancelling filter can be inserted into the control-system design in such a way that it will not bias the controller for a linear plant, and will minimally bias a controller for a nonlinear plant. This, the second part of our discussion on disturbance cancelling, describes methods to adapt the disturbance cancelling filter. The development of the algorithm is done in three stages. The final stage is the most general, but occasions have been found where one of the earlier (simpler) methods work just as well and adapt more quickly.

5.3.1 Controller Feedback

We first consider cancelling disturbance for a linear plant. Based on our heuristic discussion of the rôle of the disturbance canceller, we recognize that its function is that of an estimator coupled with a plant inverse. Supposing that we are performing straightforward inverse control (not model-reference based inverse control), we have already trained the controller to be the plant inverse. Therefore we can set $X = -C_{\text{COPY}}$. If the disturbance generating process is a Martingale process[3] then this method quickly and easily determines the optimal disturbance cancelling filter. Figure 5.9 depicts this method.

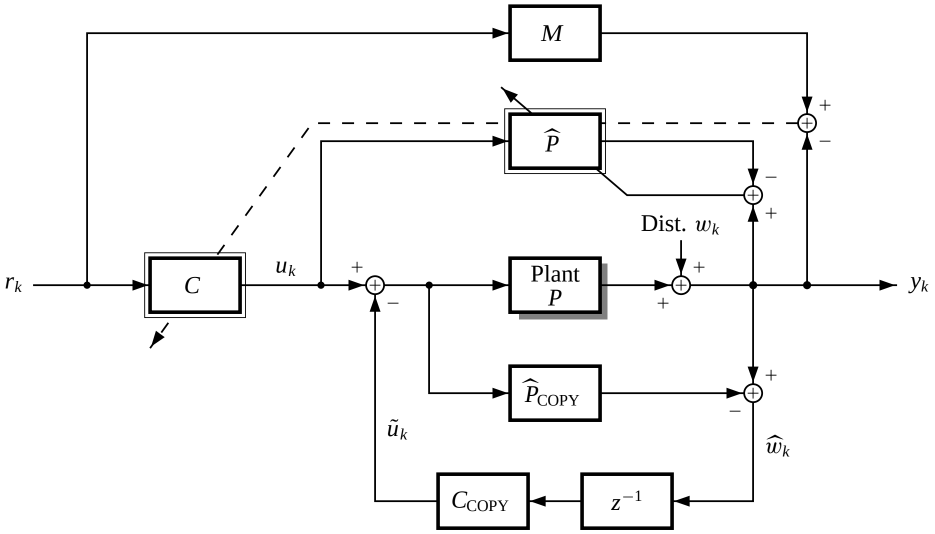


Figure 5.9: Disturbance cancelling via controller feedback.

5.3.2 Estimator Plus Controller Feedback

Still supposing that the plant is a linear system, but that the disturbance is not a Martingale process, an estimate of the disturbance is required in order to optimally cancel disturbance. Such an estimator may be implemented using an adaptive linear or nonlinear filter and is very easily trained as shown in Fig. 5.10. As has been described, the optimal estimator may be a nonlinear filter even if the plant is a linear system. This second method separates the function of the disturbance canceller very nicely so that the linear part (the plant inverse) is done by a linear filter, and the nonlinear part (the predictor) is done by a nonlinear filter.

The predictor E trains rapidly using the circuitry at the bottom of the figure since a direct error signal is available. The input to the predictor is a delayed version of the measured disturbance, $\widehat{w}_{k-2}$. The desired response for the predictor is the measured disturbance $\widehat{w}_{k-1}$. The predictor learns

[3] A Martingale process is a special case of Markov process where the expected value of the current sample given the past is equal to the most recent past sample.

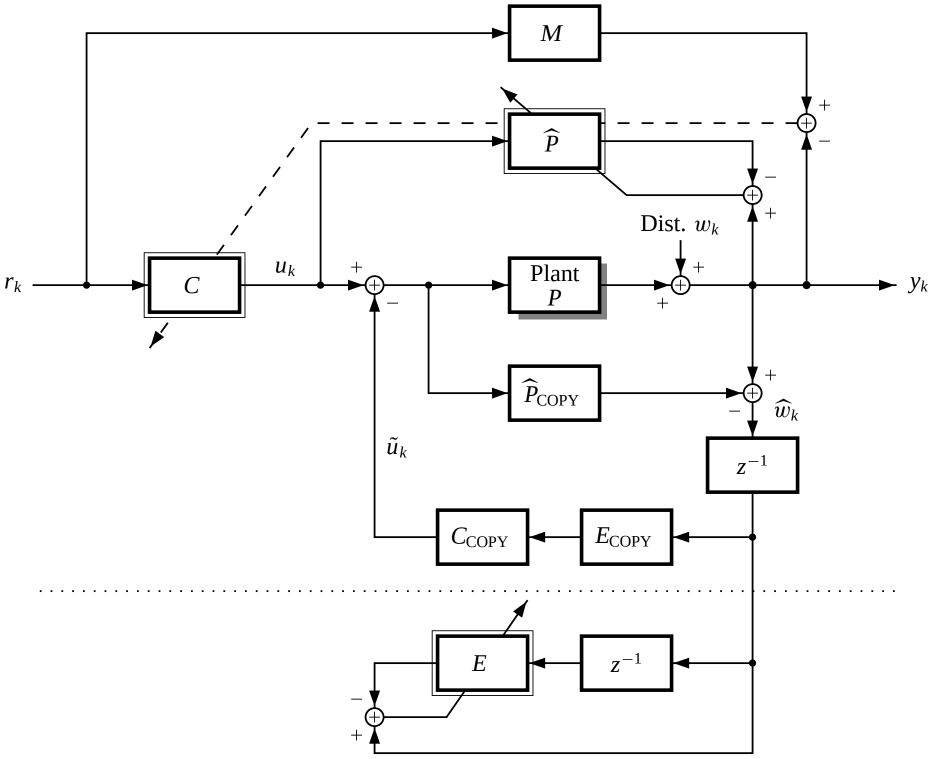


Figure 5.10: Disturbance cancelling via estimator plus controller feedback.

to predict the disturbance one time step into the future. Assuming that the disturbance is a stationary random process, a digital copy E_{COPY} of the predictor E is used to predict $\widehat{w}_k$ from $\widehat{w}_{k-1}$.

If the plant is nonminimum phase, then a delayed inverse is used as C, and the estimator must be trained to estimate disturbance the correct number of samples into the future.

5.3.3 Training X In-Place

If the plant is nonlinear, the above methods cannot be used. The third and most general method is to adapt X in-place. The method works on the following basis. We know that the system error is composed of three parts:

- One part of the system error is dependent on the input command vector $\vec{r}_k$ in C. This part of the system error is reduced by adapting C.

- Another part of the system error is dependent on the estimated disturbance vector $\vec{\widehat{w}}_k$ in X. This part of the system error is reduced by adapting X.

- The minimum-mean-squared-error. This part of the system error is independent of both the input command vector in C and the estimate disturbance vector in X. It is either irreducible (if the system dynamics prohibit improvement), or may be reduced by making the tapped-delay lines at the input to X or C larger. In any case, adaptation of the weights in X or C will not reduce the minimum-mean-squared-error.

- The fourth possible part of the system error is the part which is dependent on both the input command vector and the disturbance vector. However, by assumption, r_k and w_k are independent, so this part of the system error is zero.

Using the BPTM algorithm to reduce the system error by adapting C, as discussed in Chap. 4, will reduce the component of the system error dependent on the input r_k. Since the disturbance and minimum-mean-squared-error are independent of r_k, their presence will not bias the solution of C. The controller will learn to control the feedforward dynamics of the system, but not to cancel disturbance.

If we were to use the BPTM algorithm and backpropagate the system error through the plant model, using it to adapt X as well, the disturbance canceller would learn to reduce the component of the system error dependent on the estimated disturbance signal. The component of the system error due to unconverged C and minimum-mean-squared-error will not bias the disturbance canceller.

This method is illustrated in Fig. 5.11 where a complete integrated MIMO nonlinear control system is drawn. The plant model is adapted directly, as before. The controller is adapted by backpropagating the system error through the plant model and using the BPTM algorithm of Chap. 4. The disturbance canceller is adapted by backpropagating the system error through the *copy* of the plant model and using the BPTM algorithm as well. So we see that the BPTM algorithm serves two functions: it is able to adapt both C and X.

Using BPTM to adapt X works well for either linear or nonlinear systems. If the disturbance is not Martingale, if the plant is nonlinear or nonminimum phase, or if the controller is trained to perform model-reference control (where the reference is not a simple delay) then the “controller feedback” solution is not optimal. If the plant is nonminimum phase and a large delay is used in the controller or if the controller is trained to perform model-reference control then the “estimator plus controller feedback” solution is not optimal either. The BPTM algorithm produces the best results of the three methods outlined. Since it adapts slowly, however, it can be useful to initialize X using one of the other two methods.

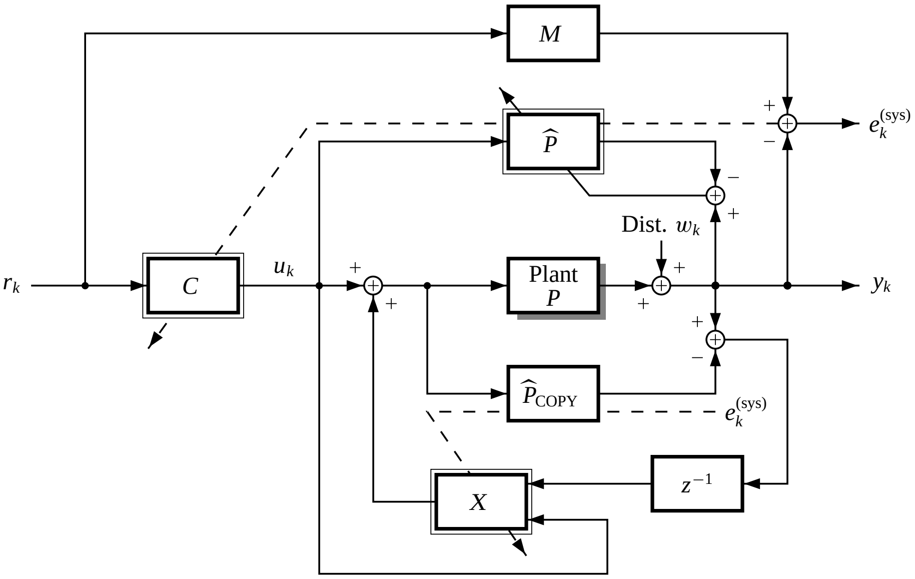


Figure 5.11: An integrated nonlinear MIMO system.

5.4 Simulation Examples

We have seen some analytical results relating to disturbance cancelling, and several methods to perform disturbance cancelling in the context of adaptive inverse control. It is now time to present some simulation results for the plants of Chap. 3 to verify the analytical results of this chapter and to demonstrate the viability of the disturbance cancelling methods.

5.4.1 Minimum-Phase Linear SISO Plant

The first examples we look at are for the minimum-phase tank of Sec. 3.2.1. We have already seen (cf. Chap. 4) simulation results showing that this plant may be very effectively controlled in a feedforward sense using either a linear FIR or nonlinear NARX filter as a controller. Controllers were adapted to perform either unconstrained control, or control where the control effort was limited to be between $5^\circ C$ and $95^\circ C$. We now look at the problem of disturbance cancelling.

The disturbance experienced by this plant was specified to be a periodic fluctuation in the temperature of the hot source. A portion of this fluctuation is coupled into the plant input, with the amount depending on the controller output u_k. As such, the disturbance is statistically dependent on the control signal. However, it can be shown that the disturbance is not correlated with u_k, and so the plant model will adapt to an unbiased solution, despite the disturbance.

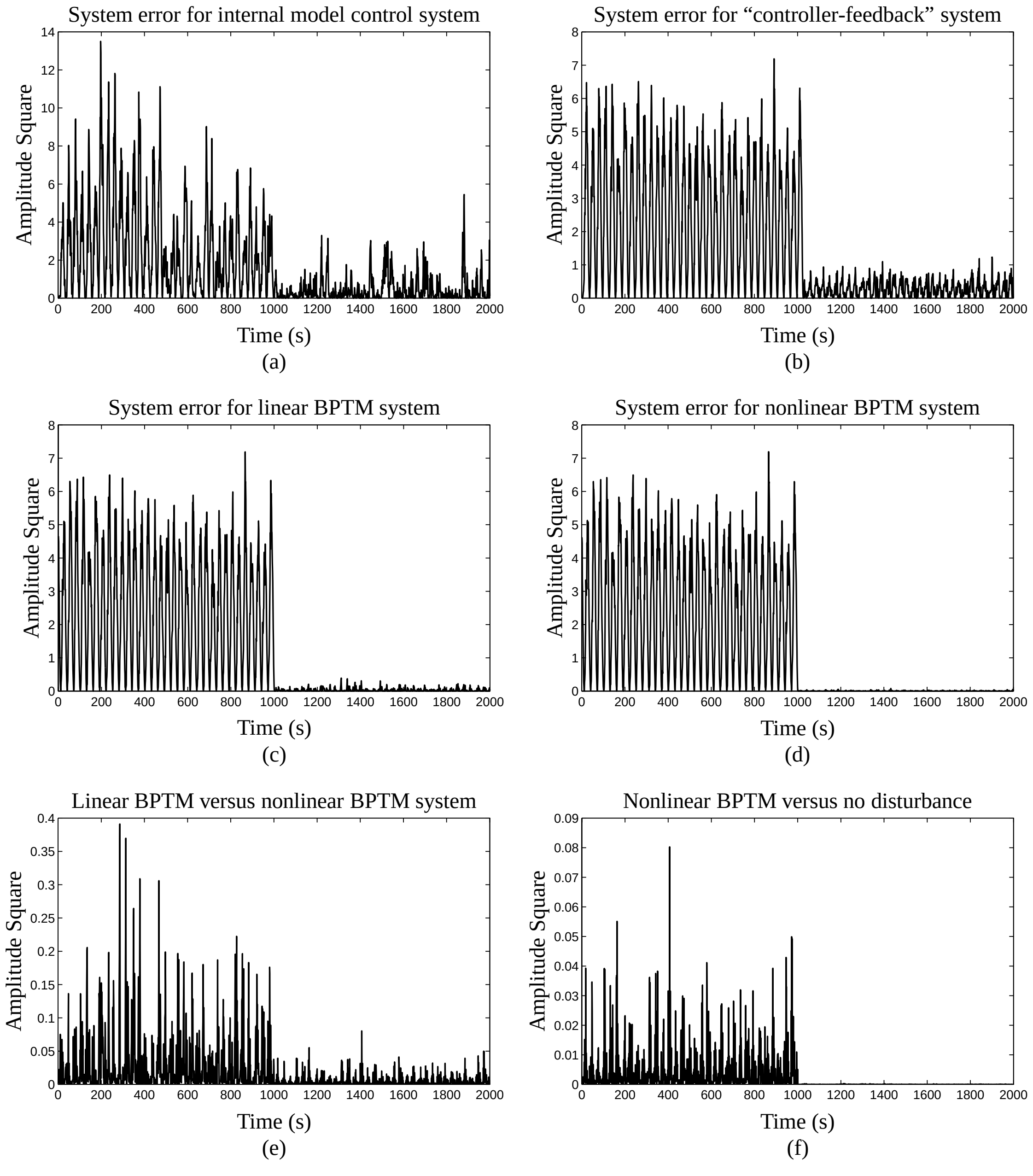
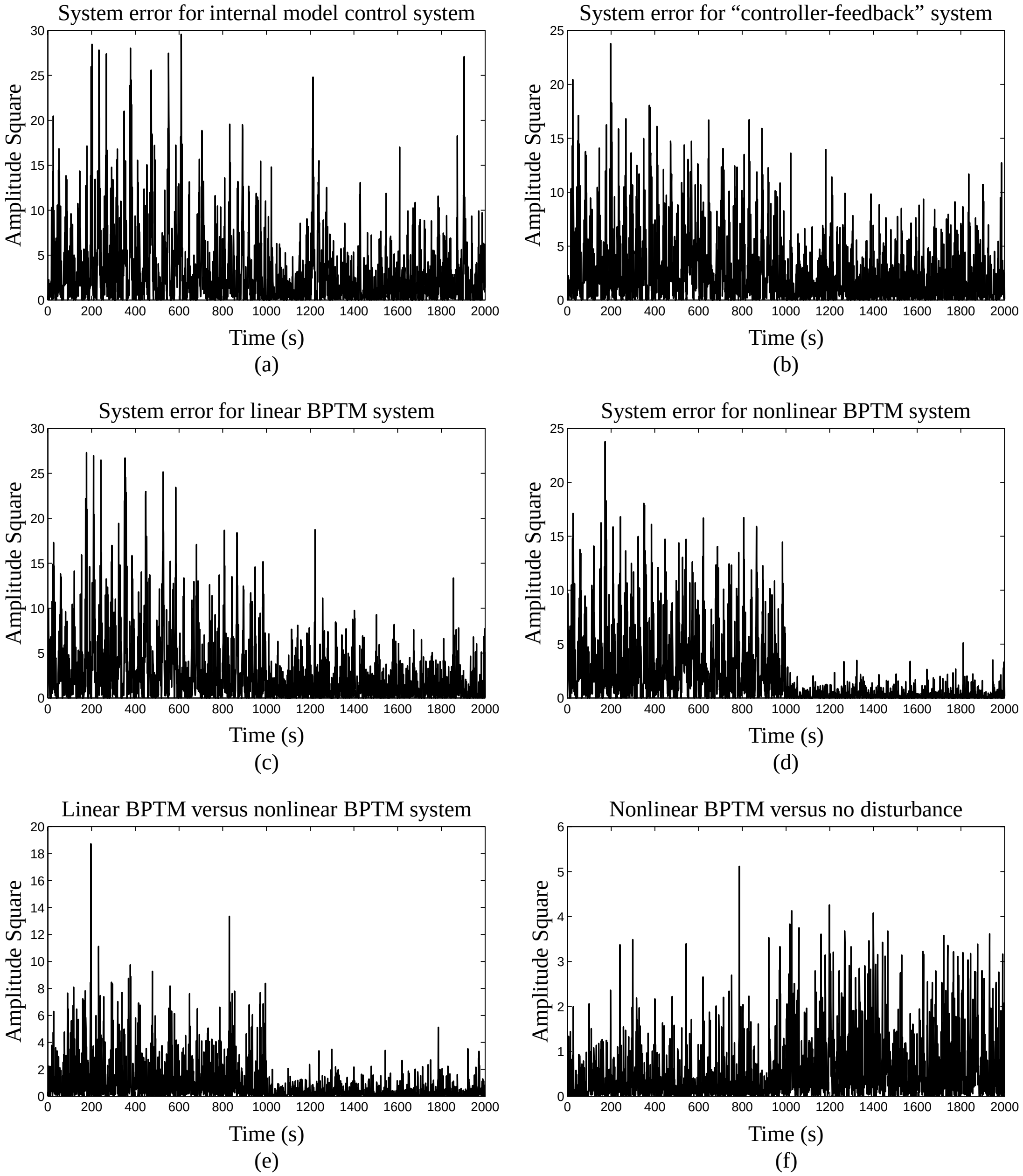


Figure 5.12: Plots showing the difference between different disturbance-cancelling schemes for the unconstrained minimum-phase linear SISO system. The disturbance canceller was turned on at time 1000.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 5.13: Plots showing the difference between different disturbance-cancelling schemes for the constrained minimum-phase linear SISO system. The disturbance canceller was turned on at time 1000.

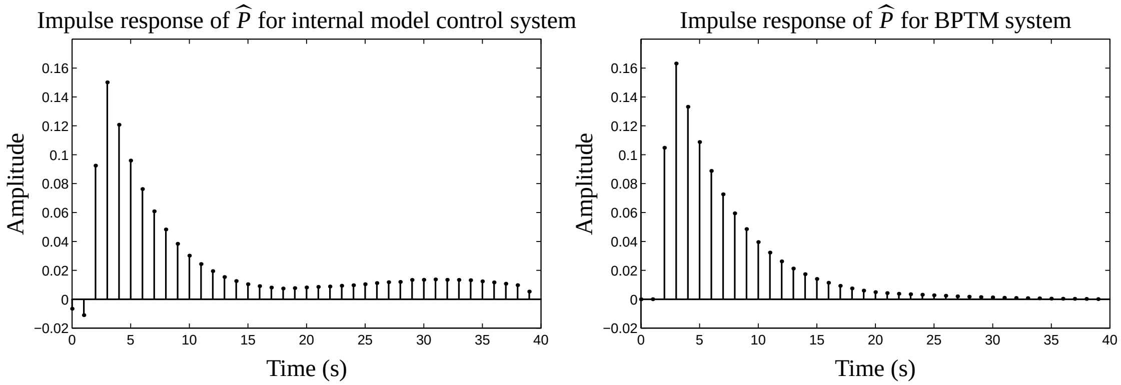
Another interesting feature of this disturbance is that it is a nonlinear random process. That is, the least mean-squared-error predictor of the current disturbance value given all previous disturbance values is a nonlinear function. We will find that the disturbance cancelling filter X which gives the best performance is therefore a nonlinear NARX filter.

Simulations were performed to adapt disturbance cancelling filters X of various types for this plant, and some results are shown in Fig. 5.12. For these simulations there were no constraints on the control effort. In the plots the system was run for 1000 seconds with the disturbance canceller turned off. Then, the disturbance canceller was turned on and the system was allowed to run for an additional 1000 seconds (the disturbance cancelling filters X had already adapted to convergence before these experiments were performed). The squared system error is plotted versus time. In Fig. 5.12(a), we see results for the internal-model-control disturbance-cancelling scheme of Fig. 5.2(b). As expected from the analysis, this system does not cancel disturbance well. The plant model, shown in Fig. 5.14(a), is biased by the disturbance. The entire system becomes biased, and the mean-squared system error is very high.

Figure 5.12(b) shows how the result improves when the system of Fig. 5.9 is used instead. This system is identical to the one in Fig. 5.2(b) if the plant model is not allowed to adapt on-line, but is the correct extension to the internal model control scheme if the plant model is allowed to adapt on-line. We see that better performance is achieved when the disturbance canceller is turned on.

Neither of these two schemes attempts to predict future disturbance values while performing disturbance cancelling. We obtain much better results in Fig. 5.12(c) when the BPTM-adapted system of Fig. 5.11 is used to perform disturbance cancelling. The disturbance cancelling filter X was a sixty-tap FIR filter whose input was the estimate of the disturbance $\widehat{w}_{k-1}$. It adapts to the solution shown in Fig. 5.18(a). The solution comprises a three-step-ahead estimator (due to the inherent two-step delay in the plant, and the one-step delay in the disturbance measurement process) convolved with a delayed plant inverse. We also see in Fig. 5.14(b) that the plant model adapted on-line using this scheme adapts to the correct solution.

The results in Fig. 5.12(d) are better yet! For these simulations, a NARX neural network filter was used as the disturbance cancelling filter. Since the disturbance process is a nonlinear process, the neural network is better able to predict future disturbance values in order to cancel them. The filter used both u_k and $\widehat{w}_{k-1}$ as input. Figure 5.12(e) compares the results using the linear and nonlinear filters. For the first 1000 seconds, the linear filter is used to cancel disturbance; for the



Impulse response of $\widehat{P}$ for internal model control system

Impulse response of $\widehat{P}$ for BPTM system

(a) (b)

Figure 5.14: Plots showing that the plant model may be biased by disturbance cancelling if on-line plant modeling is performed. (a) $\widehat{P}$ after adaptation for the internal model control system; (b) $\widehat{P}$ after adaptation for the "BPTM" system.

remaining 1000 seconds, the nonlinear filter is used to cancel disturbance. The nonlinear filter does much better.

Finally, we see in Fig. 5.12(f) that the nonlinear filter still does not cancel all of the disturbance. The system was run with disturbance and the nonlinear disturbance-cancelling filter for 1000 seconds; the disturbance (and disturbance-cancelling filter) was turned off for the remaining 1000 seconds. Not all of the disturbance can be estimated and removed.

Returning to the issue of biased plant models, Fig. 5.15 plots the mean-squared system error versus time for three different disturbance cancelling schemes. The system was allowed to run for 1×10^6 seconds without disturbance cancelling, and then the disturbance canceller was turned on. The light-gray line in Fig. 5.15 shows the mean-squared system error versus time for the internal model control system of Fig. 5.2(b). When the loop is closed, the system initially performs very well. However, as the plant model is allowed to continue adapting, it becomes biased by the disturbance. In trying to invert the plant model, the controller also becomes biased. Over time, the system error becomes worse—although still better than with no disturbance cancelling at all.

The dark-gray line shows same result for the "controller-feedback" system of Fig. 5.9. When the loop is closed, the performance is immediately better, and remains the same over time. Continued adaptation of the plant model does not degrade system performance. However, since the disturbance is not Martingale, we can do better by using the system of Fig. 5.11. The results for this system are drawn as the black line. After this disturbance canceller (whose impulse response was initialized to all zeros) was turned on, the system performance became steadily better until

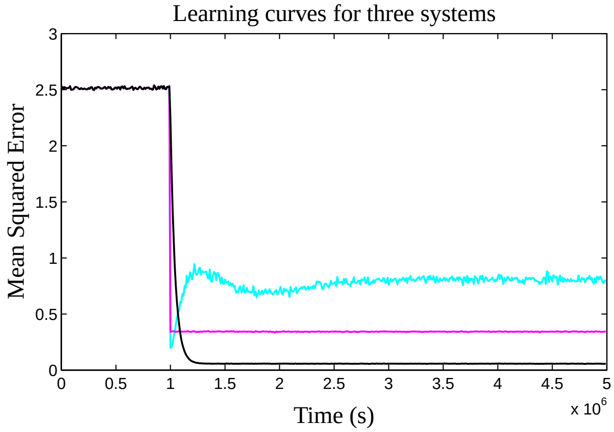


Figure 5.15: Plots showing various aspects of disturbance cancelling for the minimum-phase tank system Learning curves for the internal model control system (light gray), "controller-feedback" system (dark gray), and BPTM system (black).

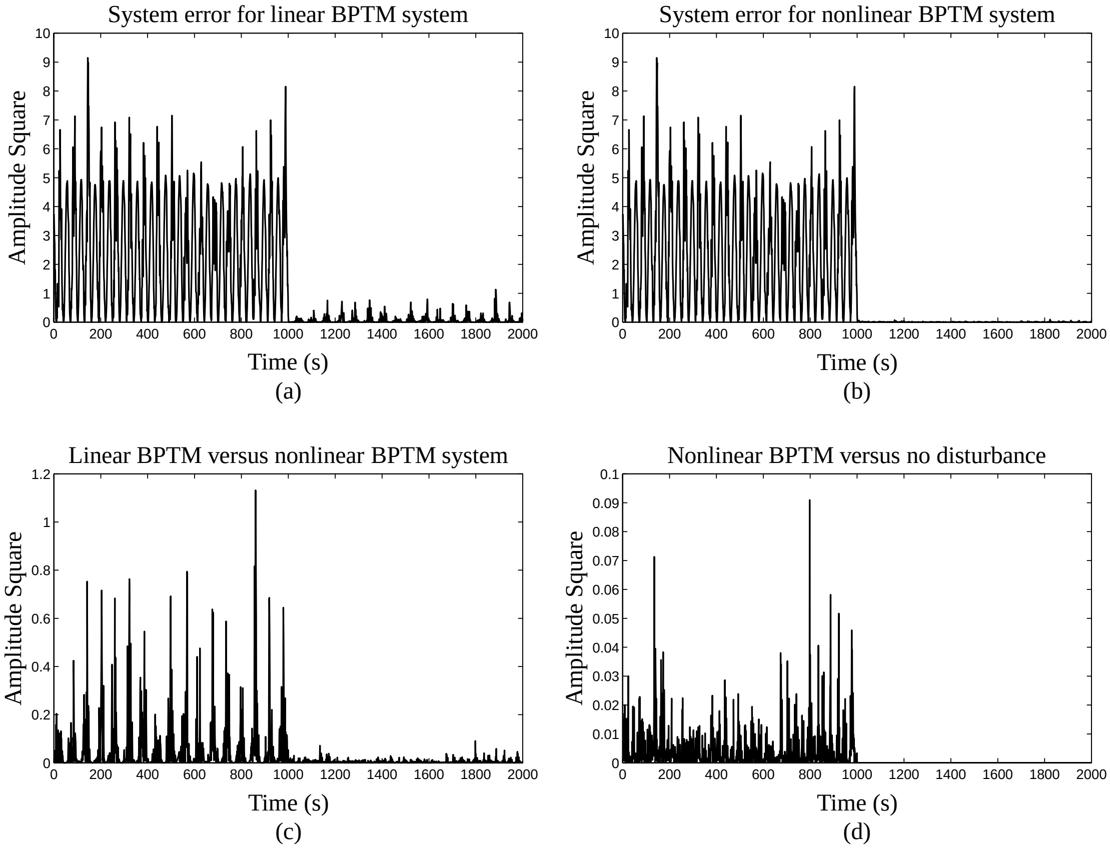
the disturbance canceller had adapted to convergence. Even better results were obtained when the disturbance canceller was allowed to be nonlinear, but the learning was slower.

Very similar simulations were performed for this plant when there were constraints on the control effort. Figure 5.13 displays the results of these simulations. We see that the squared system error is always fairly high. This is because the constraints on the control effort do not allow us to precisely control the plant. There is always a residual error due to the constraint, in addition to any disturbance which may be present.

The internal model control system results shown in Fig. 5.13(a) and the "controller-feedback" system results shown in Fig. 5.13(b) are both fairly poor. When a linear filter is used to cancel disturbance using the method of Fig. 5.11 then somewhat better results are achieved, as shown in Fig. 5.13(c). *Much* better results are achieved by using a nonlinear disturbance canceller, as shown in Fig. 5.13(d), and are compared with the system error when there is no disturbance in Fig. 5.13(f). We see that the system error with disturbance, a nonlinear disturbance canceller, and constraints on the control effort at the output of C is *lower* than the system error without disturbance but with constraints on the control effort at the output of C. The nonlinear disturbance cancelling filter is using the u_k signal to cancel disturbance and provide better tracking at the same time. This is actually cheating—the control effort at the output of C has been constrained, but the total control effort, $u_k + \tilde{u}_k$ has not been constrained. This simulation shows that the system designer must use the constrained BPTM algorithm to adapt both C and X; otherwise, the total control effort constraints may be violated.

5.4.2 Nonminimum-Phase Linear SISO Plant

Very similar simulations were performed for the nonminimum-phase tank. Since it has been determined that the internal model control system and the “controller-feedback” system do not give good performance, they were not simulated. Instead, the BPTM algorithm was used with the system of Fig. 5.11 to adapt both a linear and a nonlinear disturbance cancelling filter.



(a) (b) (c) (d)

Figure 5.16: Plots showing the difference between different disturbance-cancelling schemes for the unconstrained nonminimum-phase linear SISO system. The disturbance canceller was turned on at time 1000.

The results using the linear disturbance-cancelling filter are shown in Fig. 5.16(a). We see that it does a very credible job of removing the disturbance. However, much more disturbance remains in the system error than with the same disturbance source and the minimum-phase tank. The linear disturbance canceller is not able to predict the disturbance far enough into the future to cancel as much of the disturbance. The nonlinear disturbance canceller, on the other hand, is able to predict the future with much more accuracy. The final performance with a nonlinear disturbance canceller for the minimum phase tank and the nonminimum phase tank are very comparable.

Simulation results are plotted in Fig. 5.17 when there were constraints on the control effort. The same general comments apply as for the corresponding simulations for the minimum phase tank. Some care needs to be taken to penalize the additional control effort due to the output of X.

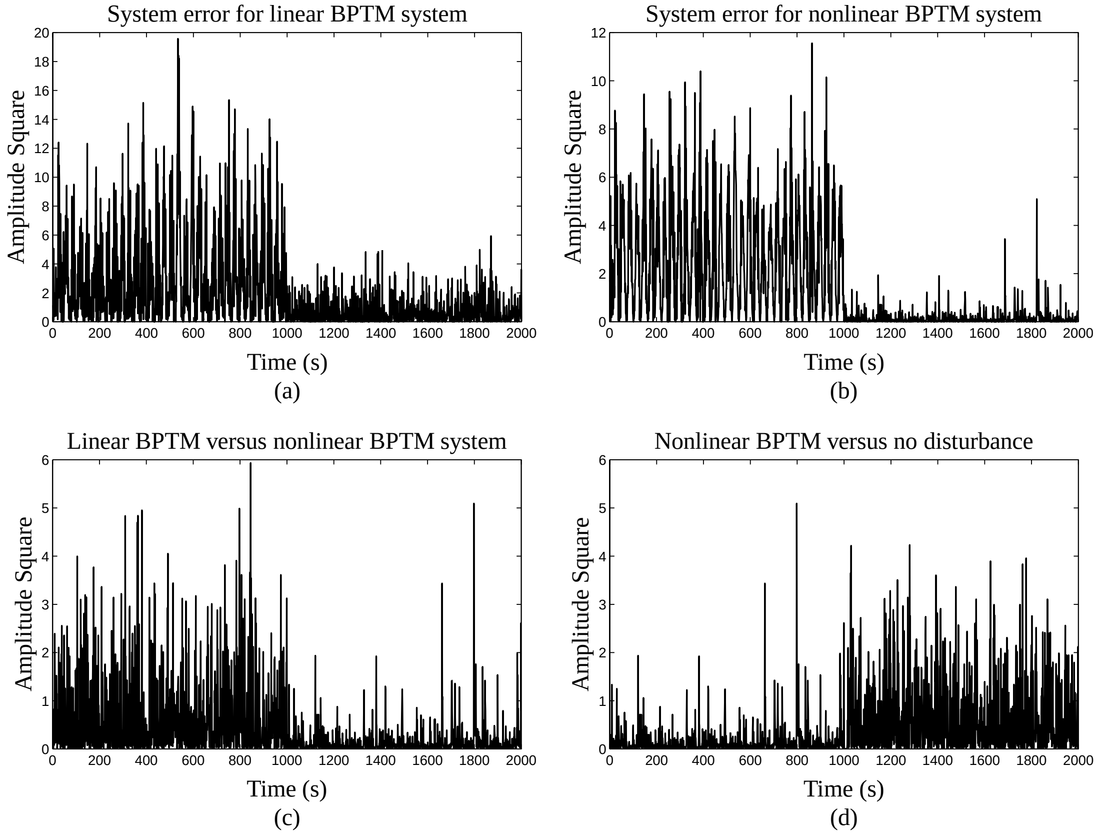
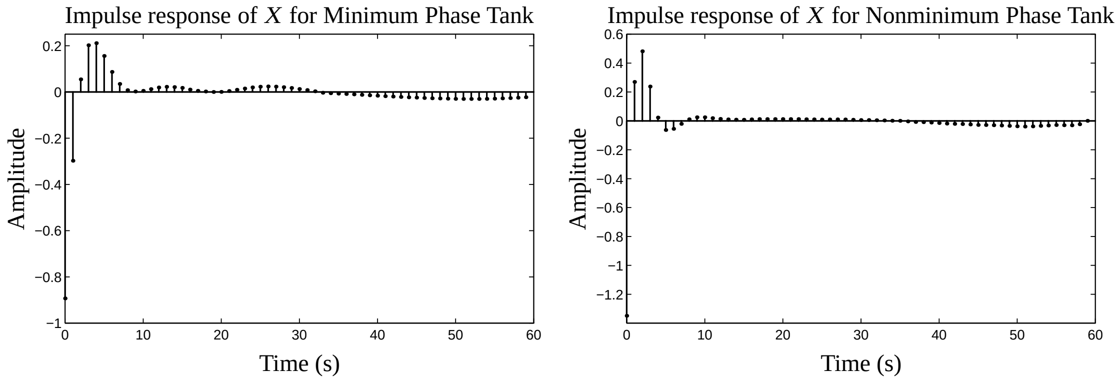


Figure 5.17: Plots showing the difference between different disturbance-cancelling schemes for the constrained nonminimum-phase linear SISO system. The disturbance canceller was turned on at time 1000.

Finally, in Fig. 5.18, the impulse responses of the linear disturbance cancelling filters for the minimum-phase and nonminimum-phase tank examples are plotted. Unlike the controllers C, the disturbance cancelling filters are fairly similar for these plants.



(a) (b)

Figure 5.18: Disturbance cancelling filter X. (a) For the minimum-phase tank; and (b) For the nonminimum-phase tank.

5.4.3 Linear MIMO Plant

The next example we look at is the Boeing 747 linear MIMO plant of Sec. 3.2.2. This plant may also be very effectively controlled in a feedforward sense. Controllers were adapted to perform either unconstrained control, or control where the slew-rate of the control effort was limited. We now look at the problem of disturbance cancelling, and consider only the case where control effort is not constrained.

The disturbance experienced by this plant was specified to be due to gusts of wind operating directly on the sideslip-angle—an internal state variable in the state-space representation of the system. Raw wind disturbance was simulated by passing white noise through a linear filter whose spectrum was the same as the spectrum of wind speeds. The resulting wind speed was divided by the forward speed of the airplane, and the arc-tangent of the quotient was added to the sideslip-angle. Hence, the disturbance source was a nonlinear random process. In all practicality, however, the disturbance was small enough that the arc-tangent operator was approximately linear. We will see that a linear disturbance canceller is able to do almost as well as a nonlinear disturbance canceller.

Figure 5.19 plots the results of disturbance cancelling. Since the plant is minimum-phase, the estimator-feedback system of Fig. 5.10 was used to cancel disturbance. This proved to train much more rapidly than the system of Fig. 5.11, and gave just as good results. Both the linear and nonlinear disturbance cancelling systems are able to remove almost all of the disturbance, as can be seen in Fig. 5.19(a) and (b). When the two are compared in Fig. 5.19(c), we can not detect any difference visually (although the results in Table. 5.1 show that the nonlinear system is able to do somewhat better). Finally, when the nonlinear disturbance-cancelling results are compared to feedforward control without disturbance in Fig. 5.19(d), we see that almost all of the disturbance has been removed.

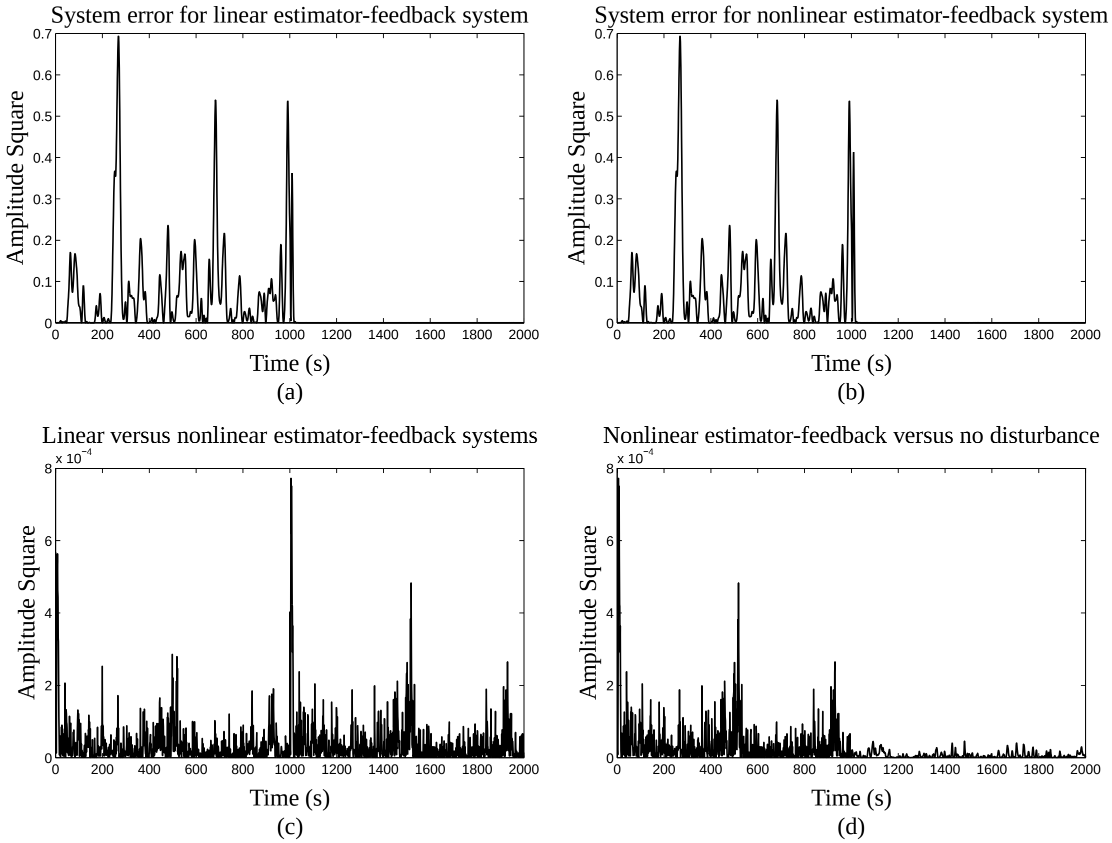


Figure 5.19: Plots showing the difference between different disturbance-cancelling schemes. The disturbance canceller was turned on at time 1000.

5.4.4 Nonlinear Plants

Simulations were also performed to demonstrate disturbance cancelling for the nonlinear plants of Chap. 3. The reader may recall that the nonlinear SISO plant was selected to be a large ocean-going ship for which we would like to control the heading angle. The nonlinear MIMO plant was a two-link robot arm for which we would like to control the joint angles. Both of these plants were initially unstable, and were stabilized using feedback. Simple unity feedback was used to stabilize the ship, and a PD controller was used to stabilize the robot.

This feedback has two effects. Most importantly, it stabilizes the system dynamics. Secondly, however, it also performs some disturbance rejection. The feedback works in such a way that the input command is modified to cancel any error in the output, and this error can include disturbance.

Therefore, we find that the results for disturbance cancelling for the nonlinear plants are not as spectacular as the results for the linear plants. The reason for this is that the system error with disturbance and without disturbance are not very different. The disturbance canceller improves upon the system error with disturbance, but the difference is so small that the disturbance canceller may be considered unnecessary for these plants. Results in Table. 5.1 show that the disturbance canceller does improve performance slightly.

Disturbance can not always be ignored with nonlinear plants! Simulations for nonlinear SISO plants in App. B show that the methods developed for nonlinear disturbance cancelling can work very well. The reader is referred to that appendix for further details.

5.5 Summary

This chapter discusses disturbance cancelling for linear and nonlinear, SISO and MIMO plants. It is organized into three main divisions. The first division explores some analysis of disturbance cancelling; the second division presents methods to adapt a disturbance canceller; and the third division presents results to verify the analysis and the disturbance cancelling algorithms.

The analytic section first shows that the feedforward control system of Chap. 4 still works if the plant is disturbed, but that nothing is done to reject the disturbance. That is, the output of the plant will converge to the sum of the desired output *plus* the disturbance. The conventional method to remove disturbance is to "close the loop." It is shown that if the loop is closed, on-line plant modeling will not converge to the right solution. This biases the controller as well, so the entire system performs in a sub-optimal way.

A method is adapted from reference [45] which performs disturbance cancelling in a more clever way. It is shown that a linear plant model will not be biased by this proposed scheme, but that a nonlinear plant model will still be somewhat biased. Since the plant is assumed to be stable, this bias is small, and simulations later show that the bias is acceptable.

Further analysis explores the function performed by the disturbance canceller. It is discovered that the disturbance cancelling filter X performs two duties: (1) It predicts the disturbance at the plant output one time step into the future; and (2) It computes the control signal required to eliminate that disturbance. The first duty is a function of the statistics of the disturbance signal, and the

TABLE 5.1 STEADY-STATE MEAN SQUARED SYSTEM ERROR FOR ALL PLANTS SIMULATED.

System	Feedforward Control	Int. Model Control	Controller Feedback	Linear Canceller	Nonlinear Canceller	No Disturbance
Linear SISO						
Unconstrained Minimum Phase	2.475262	0.678148	0.362881	0.032043	0.005100	0.000020
Constrained Minimum Phase	3.231202	1.697775	1.581939	1.032886	0.331095	0.757910
Unconstrained Nonminimum Phase	2.474227	—	—	0.029332	0.004689	0.000002
Constrained Nonminimum Phase	3.026145	—	—	0.587182	0.126064	0.551610
Linear MIMO						
Unconstrained	0.077286	—	—	0.000068	0.000054	0.000011
Nonlinear SISO						
Constrained Dynamics	0.001326	—	—	—	0.001302	0.001248
Nonlinear MIMO						
Unconstrained	0.006061	—	—	—	0.005221	0.003594

second duty is a function of the plant dynamics. In general, an estimator to predict future values of disturbance is a nonlinear function, so the disturbance canceller is best realized by a nonlinear adaptive filter, even if the plant dynamics are linear!

In the synthesis section of this chapter, three methods are proposed to perform disturbance cancelling. The simplest methods only work for linear plants. The first simply uses a copy of the controller as the disturbance canceller. It is optimal if the disturbance is a Martingale process and the reference model for the controller is the identity function: $M(z) = I$. The second method cascades a disturbance-predicting filter with the plant inverse and is optimal if the disturbance is not Martingale, but the reference model for the controller is still the identity function. The third method works for linear and nonlinear plants, and uses the BPTM algorithm developed in Chap. 4 to adapt the disturbance canceller weights.

The third division of the chapter presented simulations to verify the analytical results and the disturbance cancelling algorithms. Simulations were performed to test disturbance cancelling for all of the plants introduced in Chap. 3, with the conclusion that better performance was always obtained using the disturbance canceller.

The internal model control scheme was simulated in conjunction with on-line plant modeling. It was shown to fail, just as theory predicts. All three of the “correct” disturbance cancelling schemes presented in the synthesis division of this chapter were simulated as well. In general, it was found that the BPTM-adapted disturbance canceller of Fig. 5.11 worked best. A surprising result of the analysis which is verified in the simulations is that the optimum disturbance canceller may be a nonlinear system even if the plant is linear. Numerical results from the simulations are tabulated in Table. 5.1. All values are steady-state system mean-squared error, averaged over 1×10^6 simulated time steps, computed after the plant model, controller and disturbance canceller had converged. Finally, Table. 5.2 tabulates the architectures of the plant models, controllers and disturbance cancellers used in simulations in this chapter. The overall conclusion is that extremely good disturbance cancelling may be achieved.

TABLE 5.2 FILTER ARCHITECTURES USED IN THIS CHAPTER.

System	Plant Model, $\widehat{P}$	Feedforward Controller, C	Linear Disturbance Canceller, X	Nonlinear Disturbance Canceller, X
Linear SISO				
Unconstrained Minimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(20,0):1}$	$\mathrm{FIR}_{(60,0):1}$	$\mathcal{N}_{([60,60],0):30:1}$
Constrained Minimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(20,0):1}$	$\mathrm{FIR}_{(60,0):1}$	$\mathcal{N}_{([60,60],0):10:1}$
Unconstrained Nonminimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(15,0):1}$	$\mathrm{FIR}_{(60,0):1}$	$\mathcal{N}_{([60,60],0):30:1}$
Constrained Nonminimum Phase	$\mathrm{FIR}_{(40,0):1}$	$\mathrm{FIR}_{(15,0):1}$	$\mathrm{FIR}_{(60,0):1}$	$\mathcal{N}_{([60,60],0):30:1}$
Linear MIMO				
Unconstrained	$\mathrm{IIR}_{(4,3):2}$	$\mathrm{IIR}_{(4,4):2}$	$\mathrm{IIR}_{(12,8):2}$	$\mathcal{N}_{(12,2):5:2}$
Nonlinear SISO				
Constrained Dynamics	$\mathcal{N}_{(20,1):8:1}$	$\mathcal{N}_{(22,1):50:1}$	—	$\mathcal{N}_{([20,20],1):10:1}$
Nonlinear MIMO				
Unconstrained	$\mathcal{N}_{(20,0):10:2}$	$\mathcal{N}_{(11,1):20:2}$	—	$\mathcal{N}_{([4,5],1):20:2}$

Chapter 6

Imperfect Sensors

If I had any humility I would be perfect.

—Ted Turner

6.1 Introduction

All work done in adaptive inverse control to this time has made the assumption that the sensors used to measure the plant output are ideal. This may often be a realistic assumption; however, for many applications an ideal sensor would either be prohibitively expensive or altogether impossible. This chapter proposes simple modifications to the existing adaptive inverse control structure to properly compensate for non-ideal sensors.

An ideal sensor measures the exact value of a system variable or output at some point in time. An imperfect sensor adds dynamics, distortion and/or noise to the measurement process. We may consider an imperfect sensor to comprise an ideal sensor, cascaded with some added dynamics S, and corrupted by sensor noise, v_k. This is shown in Fig. 6.1. The true value of the variable being sensed is y_k. The output of the sensor is $\tilde{y}_k$. The operator S may be linear or nonlinear, and may include dynamics. The sensor noise v_k is considered to be any zero-mean and bounded random process, statistically independent of y_k.

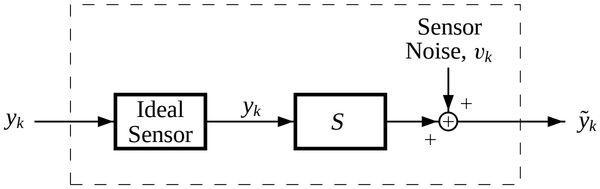


Figure 6.1: A non-ideal sensor.

In the rest of this chapter, the effect of the imperfect sensor on plant modeling, feedforward control, and disturbance cancelling are discussed. Additionally, ways are proposed to compensate for imperfect sensors, and to restore full functionality to the control system. Simulation results are presented to verify the new analysis and design.

6.2 Analysis of Adaptive Inverse Control with an Imperfect Sensor and Synthesis of a New Design

6.2.1 Effect of S on Plant Modeling

With an imperfect sensor, we must modify our perception of the plant modeling process. If an adaptive model is made in the same way as before, it now has a different meaning. We see this by examining Fig. 6.2. The sensor is explicitly included in the diagram. The "plant model" now models the dynamics of both the plant and the sensor and may include some effects due to the disturbance. If we assume that the sensor is linear (the plant may either be a linear or nonlinear dynamical system), the least mean-squared-error solution for the function performed by the adaptive block labeled $\widehat{SP}$ is

$$\begin{aligned}
\widehat{SP}^{(\text{opt})}(\vec{u}_k) &= \mathbb{E}\big[\tilde{y}_k \bigm| \vec{u}_k\big] \\
&= \mathbb{E}\big[S\big(P(\vec{u}_k)+\vec{w}_k\big)+v_k \bigm| \vec{u}_k\big] && (6.1) \\
&= \mathbb{E}\big[S\big(P(\vec{u}_k)\big) \bigm| \vec{u}_k\big]+\mathbb{E}\big[S(\vec{w}_k) \bigm| \vec{u}_k\big] && (6.2) \\
&= \mathbb{E}\big[S\big(P(\vec{u}_k)\big) \bigm| \vec{u}_k\big].
\end{aligned}$$

If, in addition to the sensor being linear, the plant is also linear, the adaptive element converges to $S(z)P(z)$. Note that if the sensor is nonlinear, then Eq. (6.2) does not follow from Eq. (6.1), and the disturbance term w_k may bias the solution for the adaptive block to include effects due to the disturbance. Zero-mean sensor noise never biases the solution as long as it is independent of the input signal u_k.

So, the adaptive plant model includes the dynamics of both the plant and the sensor. Therefore, a change in terminology is appropriate. Whereas before we considered a plant model $\widehat{P}$, we now consider a system model, $\widehat{SP}$. If the sensor is ideal ($S = 1$), then the two are equivalent. Otherwise, we must carefully consider the effects of the sensor on the control system operation.

Note that it is not wise to try to invert the sensor dynamics in order to isolate the plant dynamics for plant identification. The sensor may be nonlinear or nonminimum phase and not have an

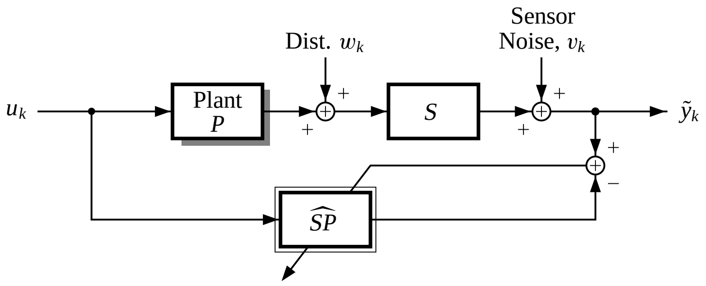


Figure 6.2: The system model is an adaptive model of the combined dynamics of S and P.

inverse. The inversion process also magnifies sensor noise. Therefore, we prefer to adapt a system model including both the sensor and plant dynamics, and modify the rest of the control scheme to compensate.

The bottom line: If the sensor noise is zero-mean, then it has no effect on plant modeling. However, the sensor dynamics do become part of the adaptive plant model. Therefore, the adaptive model is really a system model, $\widehat{SP}$, including the combined dynamics of the plant and sensor.

6.2.2 Effect of S on Feedforward Control

When used in an adaptive inverse control system, an imperfect sensor has different effects on feedforward control and on disturbance cancelling. Here, we look at the effect of the sensor on feedforward control, and how to correctly control a plant even when the sensor is not ideal. With no compensation for the imperfect sensor, a diagram of the control system is shown in Fig. 6.3.

The true output of the system is y_k. The measured output is $\tilde{y}_k$. In this figure, the measured system error $\tilde{e}_k^{(\text{sys})}$ is calculated as the difference between the desired output of the system and the measured output: $\tilde{e}_k^{(\text{sys})} = d_k - \tilde{y}_k$. This error is backpropagated through the system model via the BPTM algorithm and is used to adapt the controller. Nothing has changed in this diagram (compared with Fig. 4.9) except for the interpretation of the measured signal $\tilde{y}_k$ and the rôle of the system model. Previously, y_k was equal to $\tilde{y}_k$, and $\widehat{P}$ was modeled. Now, $\tilde{y}_k$ includes sensor dynamics and sensor noise, and $\widehat{SP}$ models the combined sensor and plant dynamics.

This system adapts C such that the mean square of $\tilde{e}_k^{(\text{sys})}$ is minimized. Thus, the measured output $\tilde{y}_k$ matches the desired output d_k as closely as possible, in a mean-square sense. Unfortunately, we do not want $\tilde{y}_k$ to match d_k; rather, we want the true output y_k to match the desired output d_k.

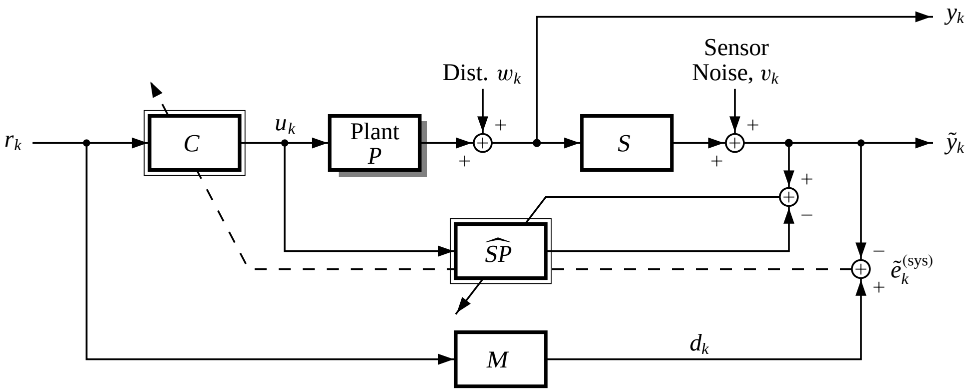


Figure 6.3: Uncompensated feedforward control.

To compensate for the imperfect sensor we must first note that the sensor noise does not affect the control as long as it is zero mean and uncorrelated with the plant input, u_k (independence is required in a nonlinear control system). Under these conditions, we have seen that the system model will converge to an unbiased solution: $\widehat{SP} \to SP$. The controller will also converge, albeit to the wrong solution: $C \to [SP]^{-1}M$, rather than the correct solution, $C \to [P]^{-1}M$. The problem with performing adaptive (feedforward) inverse control with an imperfect sensor is not due to the sensor noise, but due to the sensor dynamics.

As one approach to solving this problem, we might try to find a way to adaptively model S and P separately, and to use the models $\widehat{S}$ and $\widehat{P}$ to adapt a controller C. However, this is fundamentally impossible. The measured system output, $\tilde{y}_k$, consists of two components. One component is the part of $\tilde{y}_k$ which is correlated with the control input u_k. The other component is uncorrelated with u_k. The system model adapts to explain the part of $\tilde{y}_k$ which is correlated with u_k; the rest is a combination of disturbance and sensor noise. The correlated part is a function of two dynamical systems connected in cascade with a single vector input u_k and vector output $\tilde{y}_k$. Without prior information on S or P, we cannot separate them using this information alone.

We are forced to consider other solutions which require that either P or S (or both) be known *a priori*. The approach taken here is to assume that the sensor dynamics S are known, or approximated as $\widehat{S}$. This is felt to be realistic as manufacturers usually supply this information with their sensors. If the sensor dynamics are not known, they must be identified by some system identification method.

Our model of the sensor dynamics is called $\widehat{S}$. These are incorporated into the control system as shown in Fig. 6.4. The reference model M is now cascaded with the approximate sensor dynamics $\widehat{S}$, and the desired response signals are re-labeled. The output of the reference model is called d_k

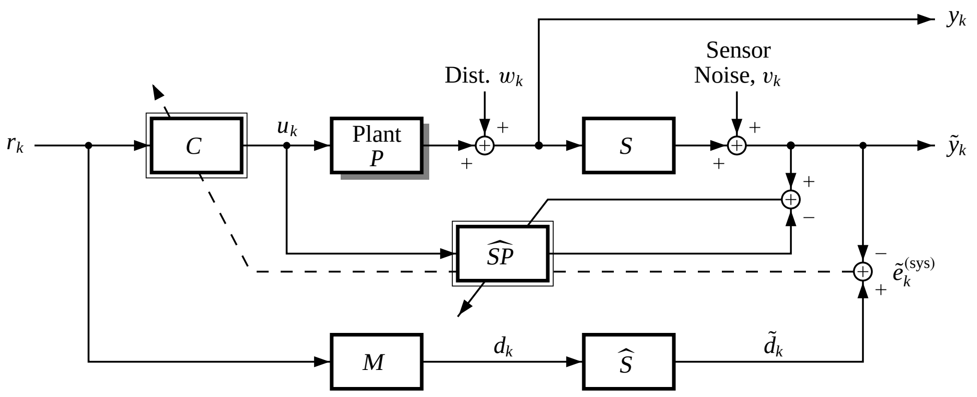


Figure 6.4: Compensated feedforward control.

and the output of the sensor model is called $\tilde{d}_k$. A "measured system error" is calculated to be $\tilde{e}_k^{(\mathrm{sys})} = \tilde{d}_k - \tilde{y}_k$. If we assume $\widehat{S} = S$ and minimize the mean square of this measured system error, we minimize the mean square of the unmeasurable system error $e_k^{(\mathrm{sys})} = d_k - y_k$. Using operator notation and the method from Sec. 5.2.2 we see that actual plant output converges to the correct plant output if the sensor is invertible.

$$\begin{aligned}
SPC &\to \widehat{S}M \\
PC &\to S^{-1}\widehat{S}M \\
&= S^{-1}\big[S + \Delta_S\big]M \\
&= M + \big[S^{-1}\Delta_S\big]M \\
&= (PC)^{(\mathrm{opt})} + \big[S^{-1}\Delta_S\big]M, \qquad (6.3)
\end{aligned}$$

where $\widehat{S} = S + \Delta_S$. So, if $\widehat{S} = S$, then the system is properly controlled. If $\widehat{S} \neq S$, then the sensitivity to errors in $\widehat{S}$ is equal to $S^{-1}\Delta_S M$. If M is made to be "small" where there are uncertainties in the transfer function of the sensor, the overall sensitivity to error will be small.

The bottom line: In order to compensate for sensor dynamics and noise in the feedforward control process, we need *a priori* knowledge of either the plant model or sensor model. The approach taken here is to assume an approximate sensor model $\widehat{S}$. Using this sensor model, a small change is made to the control system to compensate for the sensor dynamics. Sensor noise does not degrade feedforward control in any way if it is zero-mean[1] and uncorrelated with the sensor input.

[1] The mean of the sensor noise may be subtracted from the output of a sensor with a non-zero-mean sensor noise to accomplish the same level of performance for any general sensor-noise signal.

6.2.3 Effect of S on Disturbance Cancelling

Thirdly, we consider the effect of a non-ideal sensor on disturbance cancelling. First, we look at the effect of sensor dynamics; secondly, we look at the effects of sensor noise.

Effect of Sensor Dynamics on Disturbance Cancelling: The feedforward dynamics of the system were compensated by cascading M with $\widehat{S}$. Let us see what effect this has on disturbance cancelling. First, we assume that the sensor noise is zero. In Chap. 5 we saw that the disturbance canceller X was adapted so that

$$M(\vec{r}_k) = \widehat{P}\big(\vec{u}_k + X(\vec{u}_k, \vec{\widehat{w}}_{k-1})\big) + w_k.$$

If we include the sensor and sensor model, then this becomes

$$\widehat{S}\big(M(\vec{r}_k)\big) = S\left(\widehat{P}\big(\vec{u}_k + X(\vec{u}_k, \vec{\widehat{w}}_{k-1})\big) + \vec{w}_k\right).$$

So, X is adapted so that the mean-squared-measured error is minimized. We want the mean-squared true error to be minimized. If $\widehat{S} = S$, then we have achieved the desired result. Even if $\widehat{S} \neq S$, this is the best we can do. Note that if S is nonminimum phase, the disturbance cancelling will be poorer than if the sensor were ideal, since sensor dynamics must be implicitly inverted to cancel disturbance.

Effect of Sensor Noise on Disturbance Cancelling: We saw in Sec. 6.2.2 that sensor noise does not affect the feedforward dynamics of the system. We will see that it does affect the feedback dynamics. The circuitry of Fig. 5.11 for disturbance cancellation may be too aggressive if there is sensor noise. Let us consider the effects of the two uncertainty sources in the system:

- w_k: Plant disturbance is physical and affects the true output of the system. We want to cancel as much of it as possible.
- v_k: Sensor noise is not physical—it is an artifact of the measurement process and is not part of the true output of the system. We do *not* want to cancel it.

If we use the disturbance cancelling system we have developed so far, then our estimate of the plant disturbance $\widehat{w}_k \approx S(\vec{w}_k) + v_k$. If we cancel all of $\widehat{w}_k$ then the true plant output will be perturbed from the desired output by the amount $-S^{-1}(\vec{v}_k)$. This is bad!

We can look at this more formally. We wish to minimize the true mean-squared system error. The best that we can do is to adapt C and X such that

$$y_k^{(\mathrm{opt})} = \mathbb{E}\big[d_k \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big].$$

We do not have access to the true error signal, so we must find another way to adapt C and X. If we minimize the mean-squared measured error, we find the following:

$$\begin{aligned} \tilde{e}_k^{(\mathrm{sys})} &= \tilde{d}_k - \tilde{y}_k \\ &= \widehat{S}(\vec{d}_k) - \Big(S(\vec{y}_k) + v_k\Big) \\ &= \Big(\widehat{S}(\vec{d}_k) - v_k\Big) - S(\vec{y}_k). \end{aligned}$$

If we minimize the mean square of this error, we get

$$S\big(\vec{y}_k^{(\mathrm{opt})}\big) = \mathbb{E}\big[\,\widehat{S}(\vec{d}_k) \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big] - \mathbb{E}\big[v_k \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big].$$

We see that there is an extra term in this solution: $-\mathbb{E}\big[v_k \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big]$. Thus, the sensor noise biases the disturbance canceller. Under general conditions, minimizing the mean-squared measured error does not minimize the true mean-squared system error.

To achieve the correct solution, we define a *modified* error $\tilde{\varepsilon}_k$

$$\begin{aligned} \tilde{\varepsilon}_k &= \tilde{d}_k - \tilde{y}_k + \hat{v}_k \\ &= \widehat{S}(\vec{d}_k) - \Big(S(\vec{y}_k) + v_k\Big) + \hat{v}_k \\ &= \Big(\widehat{S}(\vec{d}_k) - v_k + \hat{v}_k\Big) - S(\vec{y}_k). \end{aligned}$$

If we minimize the mean square of this error, we get

$$S\big(\vec{y}_k^{(\mathrm{opt})}\big) = \mathbb{E}\big[\,\widehat{S}(\vec{d}_k) \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big] - \mathbb{E}\big[v_k - \hat{v}_k \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big].$$

If we define $\hat{v}_k = \mathbb{E}\big[v_k \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big]$, then

$$S\big(y_k^{(\mathrm{opt})}\big) = \mathbb{E}\big[\,\widehat{S}(\vec{d}_k) \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big].$$

If S is linear and $\widehat{S} = S$, then

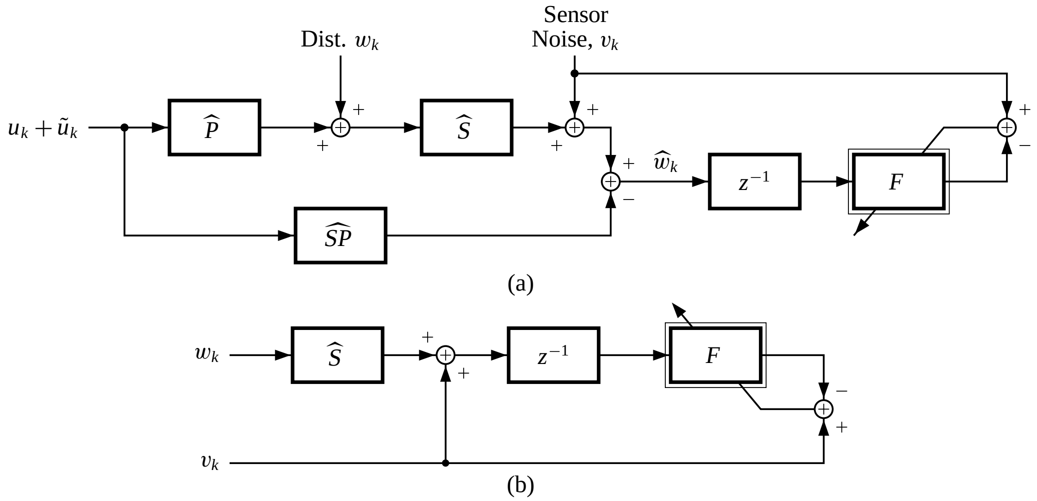
$$y_k^{(\mathrm{opt})} = \mathbb{E}\big[d_k \bigm| \vec{r}_k, \vec{\widehat{w}}_{k-1}\big],$$

which is the correct solution. Errors in $\widehat{S}$ will bias this solution somewhat. Noninvertible nonlinearities cause the true system error to be unobservable, so are also a problem (as they would be for any control system). In general, however, by minimizing the mean square of $\tilde{\varepsilon}_k$, we minimize the true mean squared system error.

To compute $\tilde{\varepsilon}_k$, we need to know $\hat{v}_k$. Again, we are faced with the lack-of-information problem. We know $\widehat{w}_{k-1} \approx S(\vec{w}_{k-1}) + v_{k-1}$, but need to separate it into its two components. We need external information to be able to do this. This is unfortunate, but unavoidable. So, using this external information, we design a filter "F":

$$\widehat{w}_{k-1} \approx S(\vec{w}_{k-1}) + v_{k-1} \longrightarrow \boxed{F} \longrightarrow \hat{v}_k$$

So, if we filter $\widehat{w}_{k-1}$, then we get $\hat{v}_k$. The filter F is designed using standard digital filtering design methods to implement the function $\mathbb{E}\big[v_k \mid \vec{r}_k, \vec{\widehat{w}}_{k-1}\big]$. Synthetic sequences of $S(\vec{w}_k)$ and v_k may be generated in a computer simulation, and a neural network or adaptive linear filter may be trained as shown in Fig. 6.5.



(a)

(b)

Figure 6.5: Generating F. If the sensor is nonlinear, the system in (a) must be used. If the sensor is linear, the system in (b) may be used.

If we know certain autocorrelation functions, we can also compute the Shannon-Bode solution for F if the sensor is linear:

$$\begin{aligned}(\phi_{\widehat{w}v})_n &= \mathbb{E}\big[\big(S(\vec{w}_{k-1}) + v_{k-1}\big)v_{k+n}\big] \\ &= \mathbb{E}\big[S(\vec{w}_{k-1})v_{k+n}\big] + \mathbb{E}\big[v_{k-1}v_{k+n}\big]\end{aligned}$$

$$\begin{aligned} &= \mathbb{E}\big[\delta_{k+1} * v_k \cdot v_{k+n}\big] \\ &= \delta_{-k-1} * (\phi_{vv})_n \\ \Phi_{\widehat{w}v}(z) &= z^{+1}\Phi_{vv}(z). \end{aligned}$$

Similarly,

$$\begin{aligned} \Phi_{\widehat{w}\widehat{w}}(z) &= S(z)S(z^{-1})\Phi_{ww}(z) + \Phi_{vv}(z) \\ F^{(\text{opt})}_{\text{causal}}(z) &= \Big[\big(z^{+1}\Phi_{vv}(z)\big)\big(\Phi^{-}_{\widehat{w}\widehat{w}}(z)\big)^{-1}\Big]_{+}\big(\Phi^{+}_{\widehat{w}\widehat{w}}(z)\big)^{-1}. \end{aligned}$$

This analytical result can be used to design the filter directly, or the spectrum may be used to derive a filter via the common digital filter design tools of least-square fitting and the Remez algorithm [28].

White Sensor Noise: An important special case is when the sensor noise v_k is white. We quickly see that $F^{(\text{opt})}_{\text{causal}}(z) = 0$. Sensor noise may be assumed to be white if the only errors are quantization noise, and a sufficient number of bits are used in the quantizer [23]. If the only sensor imperfection is quantization noise, then the structure of Fig. 5.11 works well without changes!

Function of *X* with an Imperfect Sensor: One more observation should be made before moving on to some simulation examples. Regardless of whether or not the sensor noise is correlated, proper adaptation of the disturbance canceller X will result in the internal structure shown in Fig. 6.6. It is identical in form to the structure in Fig. 5.7, except that the estimator performs a different function. Before, the estimator tried to estimate the current disturbance value given past estimates of the disturbance. Now, the estimator tries to estimate the current disturbance given past estimates of the disturbance *plus* sensor noise. Unless the estimator is able to perfectly separate the disturbance and sensor noise, it will not perform as well as previously. The gain of the disturbance canceller must be lower in order to eliminate as much disturbance as possible without trying to remove sensor noise.

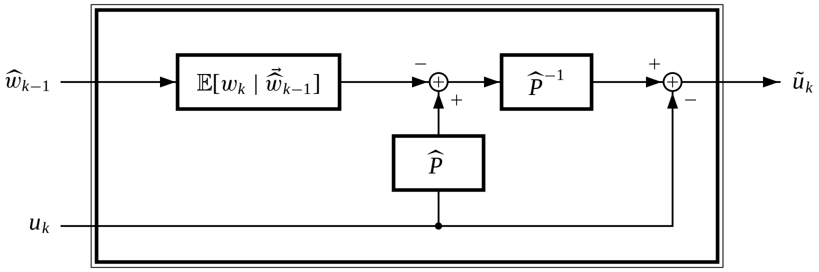


Figure 6.6: Internal structure of X if there is sensor noise. Note that $\widehat{w}_{k-1} \approx S(\vec{w}_{k-1}) + v_{k-1}$.

The bottom line: This final change in our block diagram is shown in Fig. 6.7. The overall control design has been complicated by the imperfect sensor, but each new task is easy to accomplish. The system is still very simple given the powerfully precise control which may be achieved.

6.3 Simulation Examples

We have seen some analysis demonstrating the effects of imperfect sensors on adaptive inverse control. A modified structure has been presented to compensate for the effects of the sensors. This section presents some simulation results to verify the main points of this chapter. For all of the simulations, the minimum-phase tank example from Sec. 4.4.1 was controlled. The simulations differ only in the sensor used to measure the plant output.

6.3.1 The Sensors

The sensors used in the simulations were divided into three basic categories. The first category contained a family of minimum-phase low-pass filters; the second contained a single nonminimum-phase low-pass filter; the third contained a nonlinear filter. Sensor noise was added to the output of the linear sensors.

The minimum-phase low-pass sensor was nominally a second-order digital Butterworth filter with cutoff frequency of 0.4 Hz. For a number of simulations, it was assumed that the actual filter was not precisely known. For these simulations, a family of eleven filters were used. They were all second-order digital Butterworth filters with cutoff frequencies between 0.35 Hz and 0.45 Hz, incremented by 0.01 Hz. The transfer functions of these filters are listed in Table. 6.1. The magnitude and phase response of these filters are plotted in Fig. 6.8(a) and (b). The solid line represents the nominal sensor dynamics with cutoff at 0.4 Hz. The dashed lines represent the other family members. In Fig. 6.8(c), the pole-zero plot of the family of filters is depicted. We see that all the filters are minimum-phase, with their poles safely within the unit circle, and the two zeros right on the unit circle at $z = -1$.

The nonminimum-phase low-pass sensor had the same poles as the nominal minimum-phase sensor, but had zeros at $z = -1.05$ and $z = -0.95$. Finally, the nonlinear sensor computed the function $\tilde{y}_k = 100 \tanh(y_k/c)$, where c was either 90, 100, or 110.

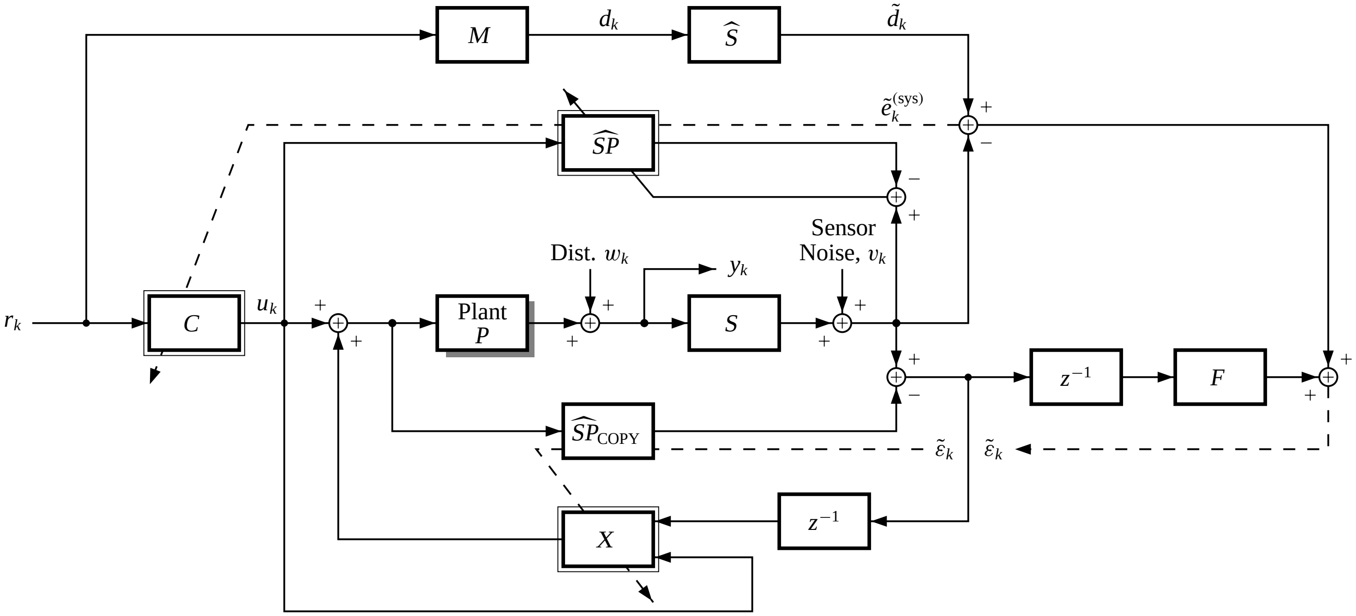
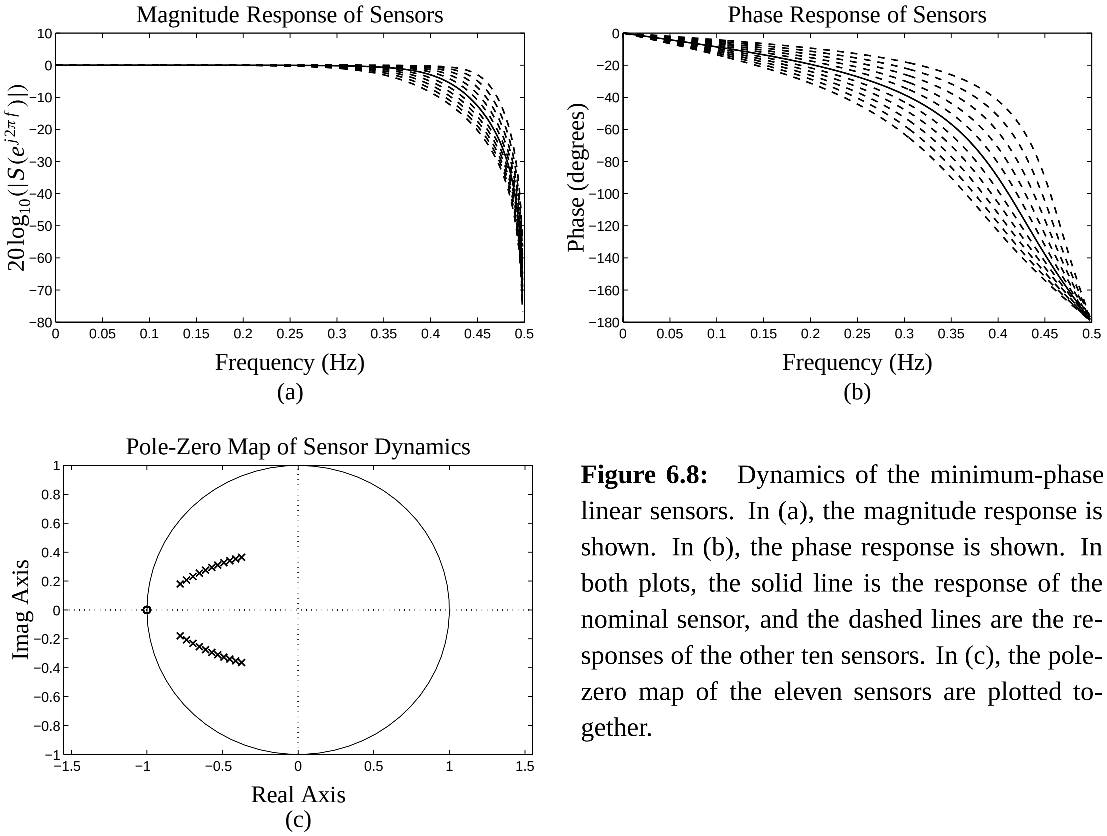


Figure 6.7: Integrated, compensated control system.



(a) (b) (c)

Figure 6.8: Dynamics of the minimum-phase linear sensors. In (a), the magnitude response is shown. In (b), the phase response is shown. In both plots, the solid line is the response of the nominal sensor, and the dashed lines are the responses of the other ten sensors. In (c), the pole-zero map of the eleven sensors are plotted together.

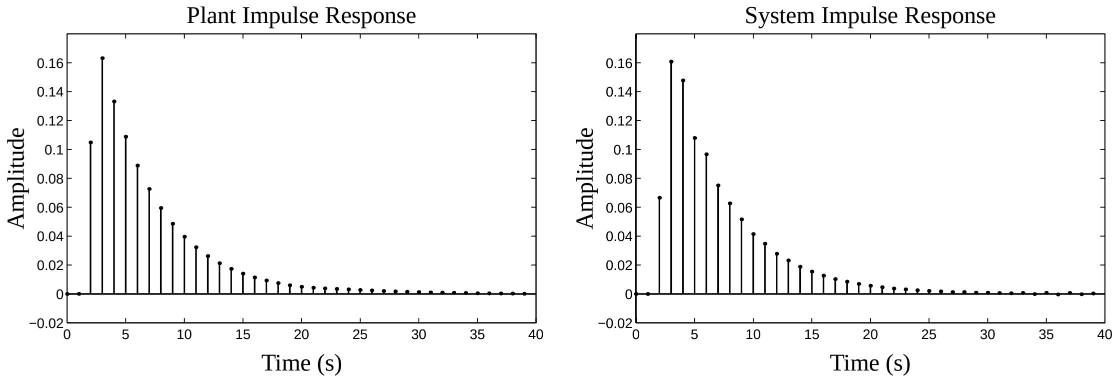
6.3.2 Linear Sensor; No Sensor Noise

The first set of simulations used the minimum-phase low-pass sensor dynamics, with no additional sensor noise. The sensor model $\widehat{S}$ was chosen to be equal to the nominal sensor. For the first run, the actual sensor S was equal to $\widehat{S}$. The system model $\widehat{SP}$ was allowed to adapt to model the combined effect of the plant and sensor dynamics. Figure 6.9 compares the impulse response of the plant and the impulse response of the entire system. Careful examination shows that there are significant differences. We can not assume that the control systems of Chap. 5 will adequately control the system. We need the modified approach derived in this chapter.

In order to test the sensitivity of the control system to the accuracy of the sensor model $\widehat{S}$, a suite of simulations were run where $\widehat{S}$ was given the nominal sensor dynamics, and the actual sensor varied in cutoff frequency from 0.35 Hz to 0.45 Hz. For each of the eleven cases, the simulation was run until convergence was achieved, and the steady-state mean squared (true) system error was recorded. Note that in practice we do not have access to the true error, but because we are

TABLE 6.1 TRANSFER FUNCTIONS OF THE MINIMUM-PHASE SENSORS. THE SUBSCRIPT DENOTES THE CUTOFF FREQUENCY OF THE FILTER IN HERTZ.

$H_{.35}(z) = \dfrac{0.5050 + 1.0100z^{-1} + 0.5050z^{-2}}{1 + 0.7478z^{-1} + 0.2722z^{-2}}$	$H_{.41}(z) = \dfrac{0.6688 + 1.3375z^{-1} + 0.6688z^{-2}}{1 + 1.2247z^{-1} + 0.4504z^{-2}}$
$H_{.36}(z) = \dfrac{0.5300 + 1.0599z^{-1} + 0.5300z^{-2}}{1 + 0.8252z^{-1} + 0.2946z^{-2}}$	$H_{.42}(z) = \dfrac{0.6998 + 1.3995z^{-1} + 0.6998z^{-2}}{1 + 1.3073z^{-1} + 0.4918z^{-2}}$
$H_{.37}(z) = \dfrac{0.5558 + 1.1116z^{-1} + 0.5558z^{-2}}{1 + 0.9034z^{-1} + 0.3197z^{-2}}$	$H_{.43}(z) = \dfrac{0.7320 + 1.4640z^{-1} + 0.7320z^{-2}}{1 + 1.3909z^{-1} + 0.5372z^{-2}}$
$H_{.38}(z) = \dfrac{0.5825 + 1.1650z^{-1} + 0.5825z^{-2}}{1 + 0.9824z^{-1} + 0.3477z^{-2}}$	$H_{.44}(z) = \dfrac{0.7656 + 1.5312z^{-1} + 0.7656z^{-2}}{1 + 1.4755z^{-1} + 0.5869z^{-2}}$
$H_{.39}(z) = \dfrac{0.6102 + 1.2204z^{-1} + 0.6102z^{-2}}{1 + 1.0622z^{-1} + 0.3786z^{-2}}$	$H_{.45}(z) = \dfrac{0.8006 + 1.6012z^{-1} + 0.8006z^{-2}}{1 + 1.5610z^{-1} + 0.6414z^{-2}}$
$H_{.40}(z) = \dfrac{0.6389 + 1.2779z^{-1} + 0.6389z^{-2}}{1 + 1.1430z^{-1} + 0.4128z^{-2}}$	



(a) (b)

Figure 6.9: Impulse response of plant compared with impulse response of system.

simulating the system, we know the true error. The results of the simulation are plotted as the solid line in Fig. 6.10.

We see that the system error is minimum where the sensor model $\widehat{S}$ is equal to the true sensor S. As S becomes more and more different from $\widehat{S}$, the mean squared system error increases. We can also see this in Fig. 6.11. In Fig. 6.11(a) and (b) we see the tracking performance of the system when the sensor has cutoff frequency at 0.35 Hz and 0.45 Hz, respectively. The tracking is good. In Fig. 6.11(c) we see tracking when the sensor model is exact. The tracking is excellent.

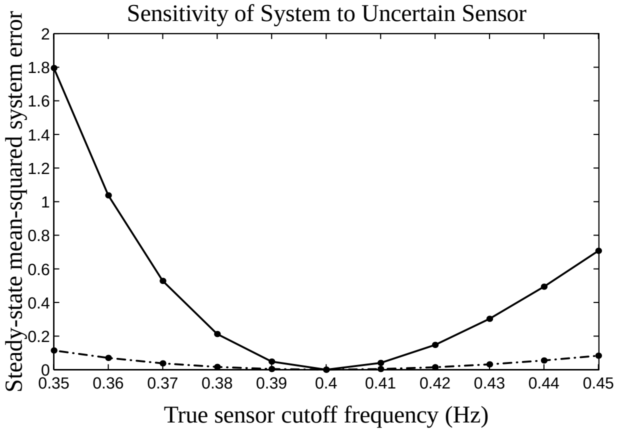


Figure 6.10: Sensitivity of system error to sensor uncertainty. The horizontal axis is the cutoff frequency of the true sensor; the vertical axis is the steady-state mean-squared system error. The solid line shows error when the reference-model is a delay of two seconds; the dashed line shows when the the reference-model is a delay of two seconds followed by a low-pass filter with cutoff of 0.25 Hz.

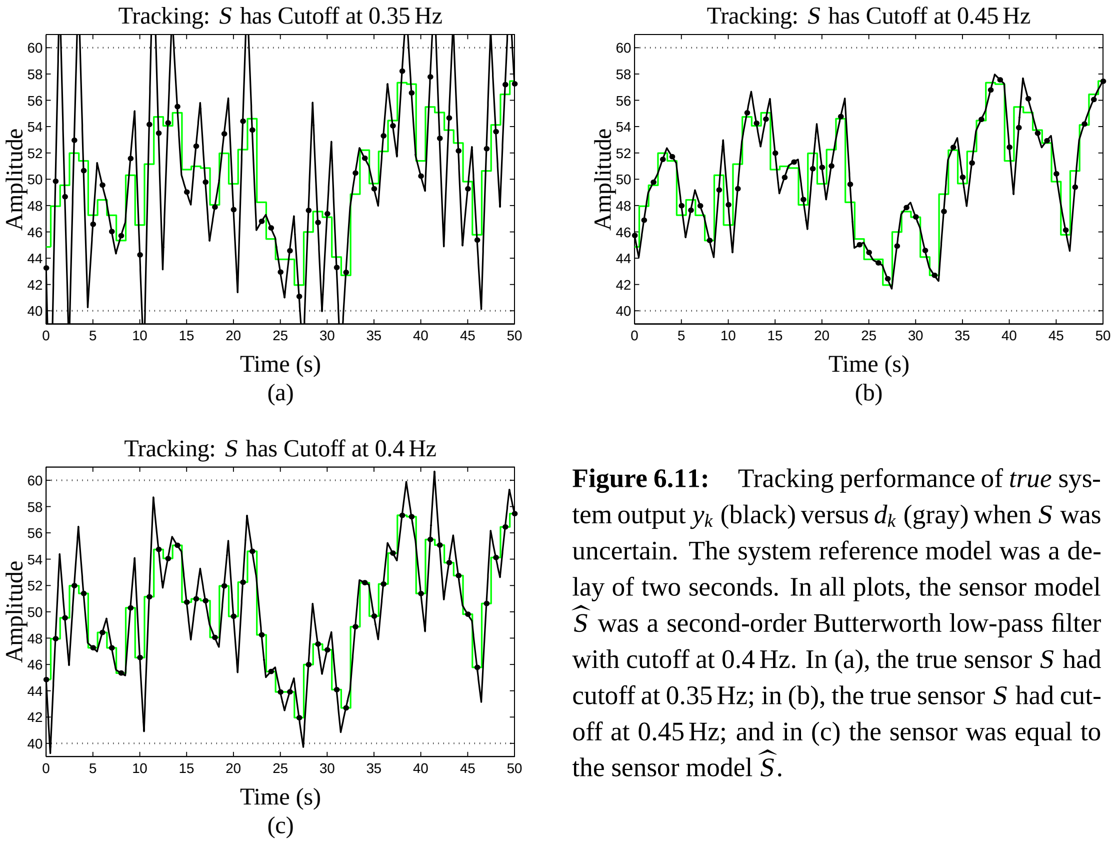
In order to decrease the sensitivity of the control system to the sensor, we consider Eq. (6.3). We see that the reference model should have low gain where the sensor has uncertainty. In order to test this, model-reference control was performed where the reference model was an fourth-order digital Butterworth filter with cutoff frequency of 0.25 Hz

$$M(z) = \frac{0.0940 + 0.3759z^{-1} + 0.5639z^{-2} + 0.3759z^{-3} + 0.0940z^{-4}}{1 + 0.4860z^{-2} + 0.0177z^{-4}}.$$

The simulations were redone, and the steady-state mean-squared (true) system error is plotted as the dashed line in Fig. 6.10. As we can see, the sensitivity of the control system to sensor uncertainty is much lower. In Fig. 6.12(a) and (b) we see the tracking performance of the system when the true sensor had cutoff frequency of 0.35 Hz and 0.45 Hz, respectively. The tracking is very good. In Fig. 6.12(c) we see the tracking when the sensor model is exact. There is not much visual difference.

Further simulations were done where there was disturbance and a disturbance-cancelling filter X was adapted. The actual sensor S was equal to the sensor model $\widehat{S}$. Table. 6.2 lists the steady-state system mean-squared error after adaptation. Comparing the results with those of Table. 5.1, we see that equivalent performance is achieved. The sensor dynamics do not affect performance.

Simulations were also done for the nonminimum-phase sensor where the sensor model was equal to the sensor. Table. 6.2 lists the steady-state system mean squared error for these simulations. Equivalent performance was achieved, compared with the case of the minimum-phase sensor. Note that it is important that the sensor model zeros and sensor zeros are identical, or an unstable inversion will implicitly take place. Since most sensors are not nonminimum-phase, this should rarely be a


Tracking: S has Cutoff at 0.35 Hz

(a)

Tracking: S has Cutoff at 0.45 Hz

(b)

Tracking: S has Cutoff at 0.4 Hz

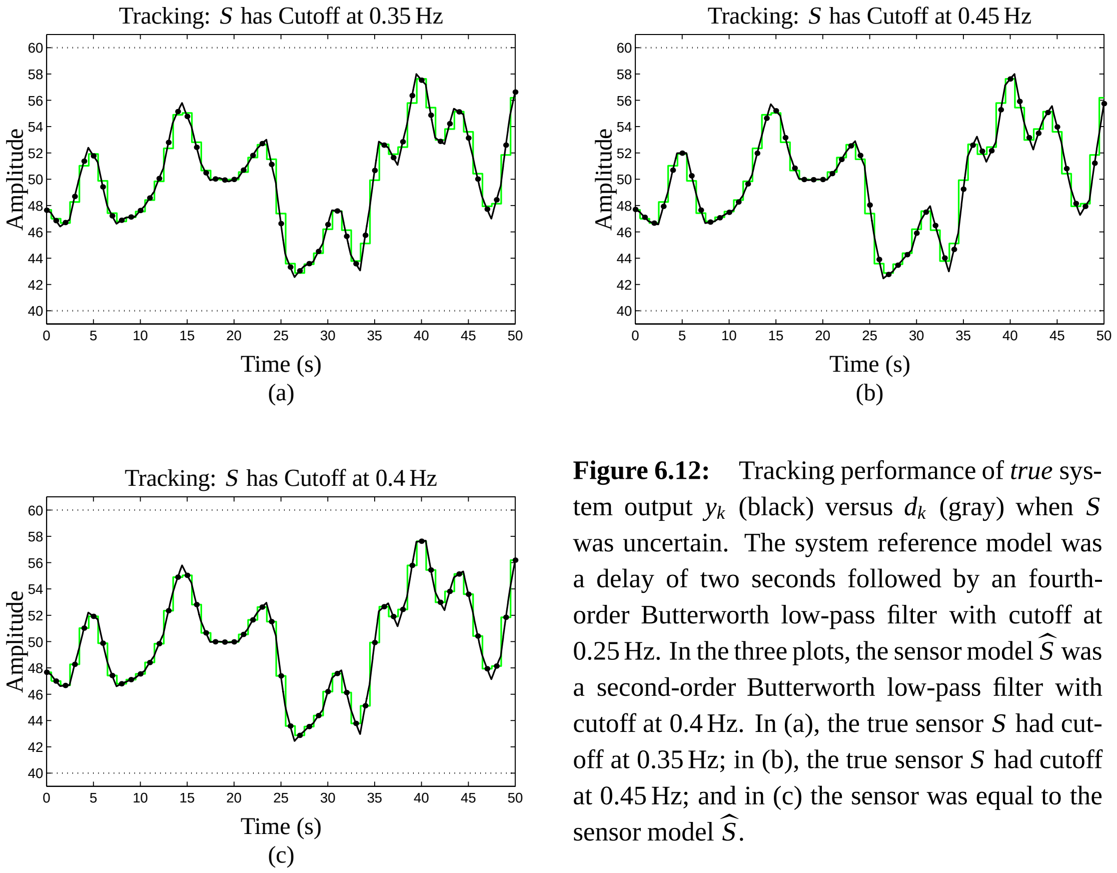
(c)

Figure 6.11: Tracking performance of *true* system output y_k (black) versus d_k (gray) when S was uncertain. The system reference model was a delay of two seconds. In all plots, the sensor model $\widehat{S}$ was a second-order Butterworth low-pass filter with cutoff at 0.4 Hz. In (a), the true sensor S had cutoff at 0.35 Hz; in (b), the true sensor S had cutoff at 0.45 Hz; and in (c) the sensor was equal to the sensor model $\widehat{S}$.

TABLE 6.2 STEADY-STATE MEAN SQUARED SYSTEM ERROR FOR DISTURBANCE CANCELLING.

Sensor	Minimum-Phase Low-Pass		Nonminimum-Phase Low-Pass	
	Linear X	Nonlinear X	Linear X	Nonlinear X
Ideal (Chapter 5)	0.032043	0.005100	0.032043	0.005100
No Sensor Noise	0.030434	0.005321	0.029315	0.005094
White Noise	0.174887	0.110836	0.136364	0.111265
Correlated Noise, No F	0.172897	—	0.170283	—
Correlated Noise, Uses F	0.057709	—	0.057171	—

problem. A sensor might be nonminimum-phase due to a pure delay term, but this is easy to measure and cancel out exactly in the sensor model.



(a)

(b)

(c)

Figure 6.12: Tracking performance of *true* system output y_k (black) versus d_k (gray) when S was uncertain. The system reference model was a delay of two seconds followed by an fourth-order Butterworth low-pass filter with cutoff at 0.25 Hz. In the three plots, the sensor model $\widehat{S}$ was a second-order Butterworth low-pass filter with cutoff at 0.4 Hz. In (a), the true sensor S had cutoff at 0.35 Hz; in (b), the true sensor S had cutoff at 0.45 Hz; and in (c) the sensor was equal to the sensor model $\widehat{S}$.

6.3.3 Linear Sensor; White Sensor Noise

Simulations were also done to test disturbance cancelling when the sensor was either the nominal minimum-phase sensor or the nonminimum-phase sensor, but when there was an addition of white sensor noise. To each measurement, a uniformly distributed random value between ±3.125 was added. This roughly corresponds to the quantization error which would be experienced if the system used a four-bit uniform quantizer when measuring y_k. As we saw earlier, the filter F is not needed when the sensor noise is white.

Results are listed in Table. 6.2. As expected, the performance is worse than if there was no sensor noise. However, the system still works well. The gain of the disturbance canceller X cannot be as high because we don't have as accurate an estimate of the true disturbance.

6.3.4 Linear Sensor; Correlated Sensor Noise

Thirdly, simulations were done to test disturbance cancelling when the sensor noise was correlated. The sensor noise was generated by passing uniformly distributed i.i.d. random variables with maximum magnitude 0.1 through a first-order Markov filter whose pole was at $z = 0.95$. Because the sensor noise is highly correlated, much of it can be removed, and the disturbance cancellation is better. Filters F were adapted for the minimum-phase and nonminimum-phase sensor using the system of Fig. 6.5(b).

Table. 6.2 lists two entries where there is correlated sensor noise. The first entry shows what happens when the correlation in the sensor noise is ignored and the filter F is not used. The steady-state system mean-squared error is high. The second entry shows the steady-state mean-squared system error if the filter F is used in the adaptation of X. The error is much lower, verifying the earlier analysis.

6.3.5 Nonlinear Sensor

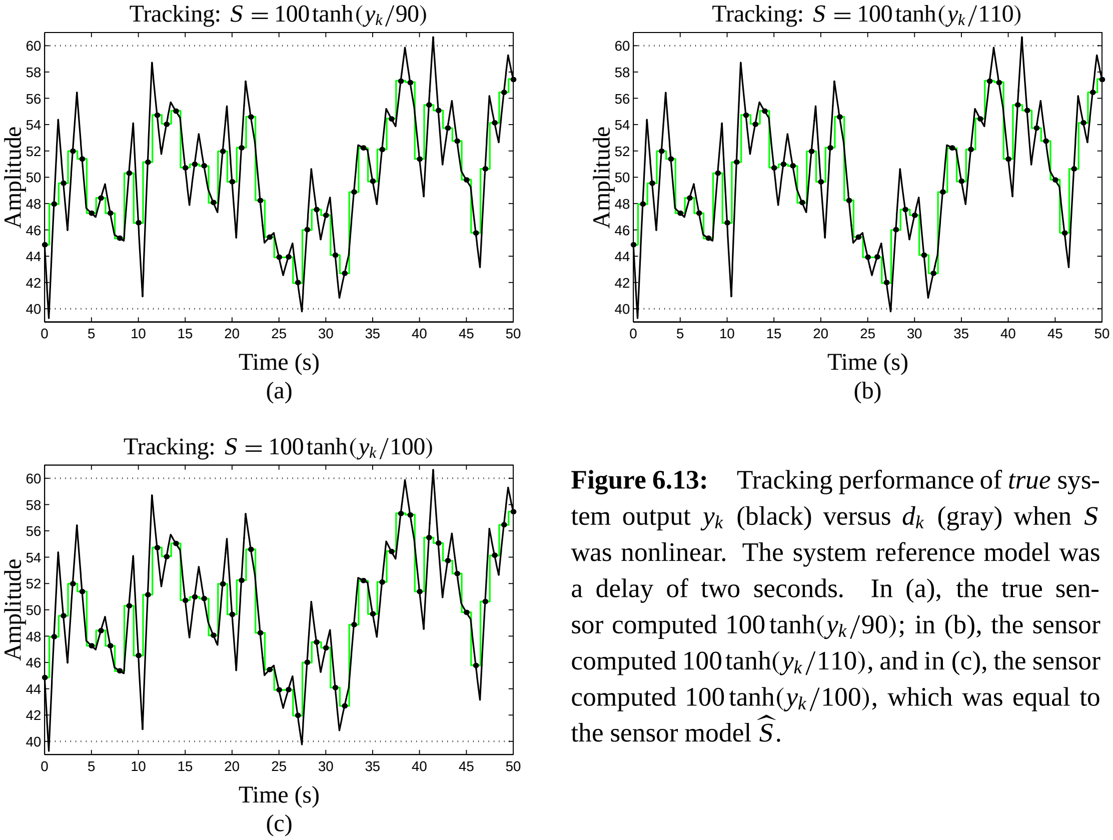
Finally, simulations were done with the nonlinear sensor. The sensor model $\widehat{S}$ was set to the nominal sensor $\widehat{S}(y_k) = 100 \tanh(y_k/c)$, with $c = 100$. The actual sensor was of the same form, with $c = 90, 100$ or 110. Tracking performance for these three cases is shown in Fig. 6.13. The tracking is very good in all three plots.

6.4 Summary

This chapter discusses the effect of non-ideal sensors on the adaptive inverse control scheme. It is organized into two main divisions. The first division analyzes the problem of non-ideal sensors and proposes a solution. The second division presents simulation results to validate this new scheme.

A non-ideal sensor was first defined. An ideal sensor measures the exact value of a system variable or output at some point in time. An imperfect sensor adds dynamics, distortion and/or noise. It was shown that non-ideal sensors have different effects on plant modeling, feedforward control and disturbance cancelling.

With an ideal sensor, an adaptive plant model converges to the undisturbed dynamics of the plant. With a non-ideal sensor, the adaptive model converges to the combined dynamics of the plant and the sensor, and may include a bias based on the disturbance if the sensor is nonlinear. Therefore, we no longer consider an adaptive plant model $\widehat{P}$, but now consider an adaptive system



(a)

(b)

(c)

Figure 6.13: Tracking performance of *true* system output y_k (black) versus d_k (gray) when S was nonlinear. The system reference model was a delay of two seconds. In (a), the true sensor computed $100\tanh(y_k/90)$; in (b), the sensor computed $100\tanh(y_k/110)$, and in (c), the sensor computed $100\tanh(y_k/100)$, which was equal to the sensor model $\widehat{S}$.

model $\widehat{SP}$ which contains the dynamics of both the plant and the sensor. Sensor noise does not bias the adaptive system model if it is zero-mean and independent of the true signal being sensed.

If the feedforward-control-system architecture of Fig. 6.3 were used with a non-ideal sensor, the controller would converge to the wrong solution. The measured output $\tilde{y}_k$ would track the desired output d_k very well, but the true system output y_k would not. The control system architecture must be revised to compensate for the sensor dynamics. The revised architecture is shown in Fig. 6.4. The reference-model M is now cascaded with an estimate of the sensor dynamics $\widehat{S}$. If $\widehat{S} = S$ then minimizing the mean-squared measured system error also minimizes the mean-squared true system error. If $\widehat{S} \neq S$, then the system exhibits some sensitivity to the uncertainty in $\widehat{S}$. This can be minimized by making the gain of the reference-model M low in the frequency band of uncertainty. Once again, the effect of a non-ideal sensor on feedforward control is entirely due to the sensor dynamics—zero-mean sensor noise (independent of the true signal being sensed) does not bias the solution.

Unlike system modeling and feedforward control, disturbance cancelling is not affected by the sensor dynamics. It may be affected by sensor noise. The disturbance-cancelling scheme of Chap. 5 may be too aggressive. It attempts to remove all of the measured disturbance, which now includes sensor noise. The sensor noise is not a real plant disturbance, but rather an artifact of the measurement process. We do not wish to cancel it from the plant output. If the sensor noise is correlated, then a special filter F must be included in the design to properly generate the signal used to adapt the disturbance canceller X (cf. Fig. 6.7). If the sensor noise is white, then this filter is not needed, and then the system architecture presented in Fig. 5.11 will work (if augmented with $\widehat{S}$ after the reference model). In either case, the disturbance canceller adapts to a solution which depends on the sensor noise. Since it is not able to separate the disturbance and sensor noise completely, it must have a lower gain than if the sensor were ideal. Disturbance cancelling cannot be as good as before.

The second division of the chapter presented simulations to verify the analysis of the chapter. Tests were done to show the sensitivity of the control system to uncertainties in the plant model. The sensitivity was reasonable, and was shown to decrease when a reference model M was used which had low gain where there were uncertainties in the sensor transfer function.

Tests were also done with a nonlinear sensor and it was shown that the compensated system performed very well, and was highly insensitive to uncertainty in the nonlinearity.

Tests were done without sensor noise to show that the compensated system architecture provides equivalent performance to a system where the sensor was ideal. Tests were done with sensor noise to show that system performance degrades gracefully. If the sensor noise is correlated and the filter F is not used to adapt the disturbance cancelling filter X, then performance was shown to be worse than if F is used, as theory predicted.

Finally, Table. 6.3 lists the architectures of the adaptive and non-adaptive filters used in the simulations of this chapter. The overall conclusion is that it is possible to effectively compensate for an imperfect sensor. Very good performance may be achieved.

TABLE 6.3 FILTER ARCHITECTURES USED IN THIS CHAPTER.

Adaptive Filter	Architecture
M	$\mathrm{IIR}_{(5,4):1}$
$\widehat{S}$	$\mathrm{IIR}_{(3,2):1}$
F	$\mathrm{FIR}_{(50,0):1}$
$\widehat{SP}$	$\mathrm{FIR}_{(40,0):1}$
C	$\mathrm{FIR}_{(20,0):1}$
X (linear)	$\mathrm{FIR}_{(60,0):1}$
X (nonlinear)	$\mathcal{N}_{(60,60):30:1}$

Chapter 7

Conclusions and Future Work

I hate quotations.

— Ralph Waldo Emerson

7.1 Summary

The problem of controlling a plant may be broken down into three separate tasks: stabilization of the plant dynamics; control of plant dynamics; and control of plant disturbance. Using conventional control techniques, one uses feedback to treat all three problems simultaneously. Compromises are necessary to achieve good solutions.

Adaptive inverse control is a method to treat the three control tasks separately. First, the plant is stabilized; secondly, the plant is controlled using a feedforward controller; thirdly, a disturbance canceller is used to reject plant disturbances. Adaptive filters are used as controller and disturbance canceller, and algorithms adapt the transfer functions of the filter to achieve excellent control.

Prior work in adaptive inverse control has focused primarily on feedforward control and disturbance cancelling for SISO linear plants, and on feedforward control for SISO nonlinear plants. This dissertation extends the prior work to encompass constrained feedforward control and disturbance cancelling for SISO or MIMO, linear or nonlinear plants, where the sensors used to measure the plant output are possible non-ideal.

7.1.1 Constrained Adaptive Feedforward Control

We assume that the plant is stable. If it is not stable, we must first stabilize it using feedback. Adaptive inverse control is used to control the stabilized plant.

Our next task is to make an adaptive plant model. This process was briefly outlined in Chap. 2. We assume that the adaptive plant modeling task continues while the plant is operating, so that any time-variations in the plant dynamics are learned, and so that the controller learns to control the plant as it varies.

Thirdly, we need to train a feedforward controller for the plant. This task is well understood for SISO linear plants [45] and has been studied for SISO nonlinear plants [2]. In this dissertation, we see how to extend adaptive inverse control to be able to control MIMO linear and nonlinear plants, and to satisfy constraints on the control effort.

Precision of control comes at the cost of high control effort. If very precise control is desired, the actuator signals are very large. Problems with large control effort include (1) The actuator may not be able to respond to the control command due to its physical design, thus causing degradation in the control which is not accounted for in the design; and (2) The actuator or the system being controlled may be damaged by excessive control effort. Since this is a significant problem, a method is devised to perform adaptive inverse control with constraints on the control effort.

A gradient-descent based algorithm was developed to update the weights of the controller. The algorithm decouples nicely, allowing separate implementation of the adaptive controller and plant model; only local information is needed for the weight update. Very general user-specified constraints on the control effort may be satisfied. Simulation results show that very good performance may be achieved.

If the plant is nonminimum-phase, its inverse does not exist. However, if a delay in the control action is acceptable, then a “delayed inverse” does exist, and very precise control can be performed. Choosing the correct delay, with no constraints on the control effort, is a significant problem. As the delay increases, the quality of control also increases; however, system latency is undesirable so we must compromise between precision and latency. If there are constraints on the control effort, choosing the delay is simple. There is a value of latency beyond which control precision does not improve. We choose this delay as the optimal compromise between precision of control and control latency.

7.1.2 Disturbance Cancelling

With the design completed to this point, the plant output will track the desired output if there is no disturbance. If there is disturbance, then the plant output will track a signal which is equal to the desired output plus the disturbance. For this reason, a disturbance rejection method is required.

Commonly, output feedback is used to perform disturbance rejection. An alternate method used by internal model control is to feed back an estimate of the disturbance. Neither method works for adaptive inverse control if the plant model is allowed to adapt on-line. The disturbance biases the solution of the plant model, and causes the entire system to adapt to an incorrect solution.

Instead of closing the loop directly, an adaptive filter is trained to perform disturbance cancelling. This method does not bias the plant model if the plant is linear, but does bias the model somewhat if the plant is nonlinear. Simulations seem to indicate that the bias is tolerable.

The function performed by this disturbance-cancelling filter has two parts. The first part estimates the current disturbance value given past disturbances; the second computes a signal to cancel the disturbance. Since the least mean-squared-error estimator of the disturbance may be nonlinear, a nonlinear disturbance cancelling filter is always recommended, even if the plant is linear.

The algorithm developed to adapt a feedforward controller can also be used to adapt the disturbance canceller. This is a great boon to the system designer—only one algorithm needs to be coded! Simulation results show that disturbance cancelling works very well for linear, nonlinear, SISO and MIMO plants.

7.1.3 Imperfect Sensors

One assumption which has been implicit in the above discussion is that the sensors used to measure the plant output are ideal. An ideal sensor measures the exact value of a system variable or output at some point in time. An imperfect sensor adds dynamics, distortion and/or noise. It was shown that non-ideal sensors have different effects on plant modeling, feedforward control and disturbance cancelling. If the sensors are not ideal, performance may be severely degraded.

A method is proposed to compensate for the non-ideal sensors. Simulations were done without sensor noise to show that the compensated system architecture provides equivalent performance to a system where the sensor was ideal. Tests were done with sensor noise to show that system performance degrades gracefully. An imperfect sensor may be effectively compensated, and very good performance may be achieved.

7.2 Future Work

Self-Stabilization of Unknown, Unstable Plants: To make use of inverse control, the plant must be stable or it must be stabilized. At the moment, this is the only step in the adaptive inverse control design process which is not done automatically. Real-time stabilization of an unknown, unstable and possibly nonlinear plant is a difficult problem, and must be investigated.

Fault Tolerance (to Catastrophic Failure): The biggest problem with catastrophic failure is plant stabilization. Combinations of the available parameters must be tried rapidly in order to determine if stabilization is possible, as above.

As one further step toward fault tolerance, banks of controllers might be pre-trained. There will be a controller for each of a number of possible pre-specified failure modes. In the event of a true and unexpected failure, combinations of these controllers may be used to generate a control signal which will allow continued operation even when there is catastrophic damage to the plant.

Multi-Rate Control: It was seen in Chap. 4 that a discrete-time control system experiences artifacts due to the sampling circuit on the A2D converter. Ringing in the continuous-time plant output is not sensed. The controller adapts to minimize the measured mean-squared system error, and is unaware of the fact that oscillation is occurring in the plant output.

Multi-rate control might be attempted to help compensate for discretizing the control system. The plant would be controlled at a faster rate than control signals enter the controller. By supplying multiple "desired-responses" for each control input, the output of the plant may be smoothly brought from one value to another.

Integration with Higher-Level Control: This dissertation considers regulator and servo control, but does not consider terminal control. The concepts may be applied to terminal control as follows: A plant controlled by an adaptive inverse controller would become an "equivalent new plant" to be controlled by an adaptive terminal controller. The high-level systems would supply the command input to the inverse controller, which would do the low-level detailed control of the plant. The inverse control system makes the equivalent plant simpler than the actual plant, and does an optimal job of cancelling disturbance. This facilitates the high-level control process.

Faster Learning Algorithms: Training of the controller and disturbance canceller is often a very slow process. Methods of initializing a linear controller (cf. Chap. 4) speed this process up significantly. Methods still need to be developed to speed up learning for nonlinear controllers. One possible solution might be to investigate Kalman-style learning algorithms [37, 32, 33]. These algorithms have proven to be very memory intensive, so methods must be found to simplify them.

7.3 Final Comments

As a researcher, it has been very rewarding to work on this dissertation. Although careful thought brought understanding, many of the simulation and analytical results were puzzling at first and caused much curiosity. For example, a nonlinear controller and disturbance canceller may be needed for constrained control even if the plant is linear; if the only sensor imperfection is due to quantization, then we don't need to worry about it—the scheme of Fig. 5.11 works well; if the plant is nonminimum-phase, the control latency may be chosen by eye off a simple plot.

Some of the results are still puzzling and invite further work. The accurate generalization of the controllers in App. B to follow untrained reference signals is amazing. Much further research needs to be done, and some of it is outlined above. However, it is time to wrap up this part of the work before proceeding to the rest.

It is the glory of God to conceal a matter;
to search out a matter is the glory of kings.
—Proverbs 25:2

It has been fun playing king.

Appendix A

Stability Analysis of the LMS and Backpropagation Algorithms

Remember that there is nothing stable in human affairs; therefore avoid undue elation in prosperity, or undue depression in adversity.
—Socrates

A.1 Introduction

It is a major concern for adaptive control that the adaptive process converges and is stable. This appendix reviews the LMS and backpropagation algorithms, and derives approximate stability criterion for the adaptive process by imposing limits on their adaptation rates. For the adaptive linear filter, it is shown that this stability criterion results in an algorithm similar to alpha-LMS. A correspondingly normalized version of the backpropagation algorithm is derived for neural networks. Special thanks to Raymond Shen and Daniel Carbonell who helped work out some of the details.

A.2 Linear Filters and the LMS Algorithm

An adaptive FIR linear filter is shown in Fig. A.1. The input to the filter is the signal x_k, and a vector of the present and the $N-1$ most recent input samples is stored in X_k. The output of the filter is computed to be $y_k = X_k^T W_k$, where W_k is a vector of weight values (filter coefficients).

The adaptation algorithm for this filter is the ubiquitous LMS algorithm, pioneered by Widrow and Hoff [17, 44]. LMS was chosen over other adaptive algorithms such as Recursive Least Squares

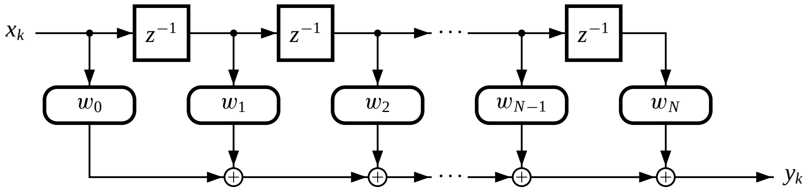


Figure A.1: Finite-impulse-response (FIR) linear filter.

(RLS), since it is more robust [16]. The weights of the filter are adapted by performing a stochastic gradient descent on the error surface of the output squared error. At any instant k, the output cost function is

$$J_k = \frac{1}{2} e_k^2,$$

where

$$e_k = d_k - y_k.$$

The corresponding gradient is:

$$\begin{aligned} \frac{\partial J_k}{\partial W_k} &= -e_k \frac{\partial y_k}{\partial W_k} \\ &= -e_k X_k. \end{aligned}$$

Therefore, the weight update rule for LMS is:

$$W_{k+1} = W_k + \mu e_k X_k,$$

where μ is the adaptation rate. The choice of μ will determine whether or not the adaptation process is stable, so we are motivated to determine its stable range. There are many ways to go about doing this, but the method chosen here generalizes well to the backpropagation case to follow.

A.2.1 Stability of the LMS Algorithm

One very powerful approach to determine stability and convergence is to use the Lyapunov method. If a time function is positive and decreasing, then it converges and is stable. There are two commonly chosen candidates for convergence—one of them is convergence of the weight vector, and the other is convergence of the output error. Convergence of the weight vector is most important for applications such as LPC speech coding which make explicit use of the weights. We are more concerned here with convergence of the output error. Therefore, the Lyapunov function V_k is:

$$V_k = \frac{1}{2} e_k^2.$$

If V_k can be shown to be positive and decreasing when the filter is adapted via the LMS algorithm, then we conclude that the output error converges and the adpative process is stable. By construction, V_k is positive. We now proceed to find conditions which ensure that V_k is decreasing.

The difference in the Lyapunov function from time step k to time step $k+1$ is:

$$\Delta_{V_k} = V_{k+1} - V_k = \frac{1}{2}\left[e_{k+1}^2 - e_k^2\right].$$

The error difference due to the learning can be represented by

$$e_{k+1} = e_k + \Delta_{e_k}.$$

Assuming that the input and desired responses remain the same from time step k to $k+1$,

$$\begin{aligned}
e_{k+1} &= d_k - y_{k+1} \\
&= d_k - W_{k+1}^T X_k \\
e_k &= d_k - y_k \\
&= d_k - W_k^T X_k \\
\Delta_{e_k} &= e_{k+1} - e_k \\
&= -\left(W_{k+1}^T - W_k^T\right) X_k \\
&= -\Delta_{W_k}^T X_k \\
&= -\mu e_k X_k^T X_k,
\end{aligned}$$

where Δ_{W_k} is determined by the LMS weight update equation. Therefore,

$$\begin{aligned}
\Delta_{V_k} &= \frac{1}{2}\left(2e_k\Delta_{e_k} + \Delta_{e_k}^2\right) \\
&= \Delta_{e_k}\left(e_k + \frac{1}{2}\Delta_{e_k}\right) \\
&= -\mu e_k \|X_k\|^2 \left(e_k - \frac{1}{2}\mu e_k \|X_k\|^2\right) \\
&= -\mu e_k^2 \|X_k\|^2 \left(1 - \frac{1}{2}\mu \|X_k\|^2\right).
\end{aligned}$$

For V_k to be decreasing, Δ_{V_k} must be negative. We find that

$$0 < \mu < \frac{2}{\|X_k\|^2}.$$

It is interesting to note that if μ is allowed to vary with the input, and (more specifically) is set to $\alpha / \|X_k\|^2$, then the above condition specifies that $0 < \alpha < 2$. This is the well-known algorithm and stability condition for alpha-LMS! It is at first surprising that we came up with this nice answer given the questionable assumption made in the derivation. However, the result helps explain the analysis being performed here.

Rather than being a steepest descent adaptation rule, alpha-LMS is known to be an error correction rule [43]. By choosing $\alpha = 0$, no part of the error is being corrected by this weight adaptation. Furthermore, by choosing $\alpha = 1$, all of the current error is being corrected, and by choosing $\alpha = 2$, we are exactly over-correcting the error. If $\alpha > 2$ then we have over-corrected the weight vector to the extent that the error increases over what it currently is. Since exactly correcting *this* error can cause other input patterns to have higher error than before (and hence cause high missadjustment in the adapted process), α is usually chosen to be between 0.1 and 1.

By relating alpha-LMS to the above result we see that the "proof" given does not explicitly promise convergence of the output but rather gives us is a sensible range for μ. We can use it to dynamically change μ, as in alpha-LMS:

$$\mu_k = \frac{\alpha}{\|X_k\|^2},$$

or, we can make μ constant over time by choosing:

$$\mu = \inf_{0 \le j < \infty} \frac{1}{\|X_j\|^2},$$

or, we can use the minimum μ computed to this point in time:

$$\mu_k = \min_{0 \le j < k} \frac{1}{\|X_j\|^2}.$$

The adaptive learning rate of the later method is the one used in this work. It ensures that we are never over-correcting the error. Furthermore, the initial high learning rate allows faster convergence.

A.3 Neural Networks and the Backpropagation Algorithm

Neural networks are interconnected structures of simple processing elements which crudely model the function of a biological neuron. Each artificial neuron (hereafter referred to simply as neuron) has the composition shown in Fig. A.2. Internally, the scalar product of the input vector[1] X_k

[1] The input vector is augmented by adding a zeroth element, always equal to 1.

and a weight vector W_k is computed, and the output is a nonlinear function of this scalar product: $y_k = f(X_k^T W_k)$. In this work, the nonlinear function is chosen to be the sigmoidal function: $f(\cdot) = \tanh(\cdot)$. Modifications to the value of the weight vector allow different output functions to be realized.

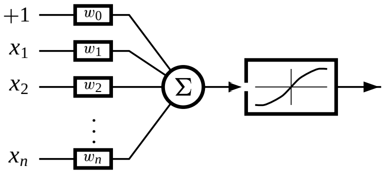


Figure A.2: Artificial neuron.

By combining many of these simple neurons into a layered network, where the input vector of a given neuron is comprised of all the outputs of the neurons in the previous layer, a very powerful computational tool is achieved. The layer of neurons connected to the output of the network is called the output layer, and all other layers are called hidden layers. This structure is shown in Fig. A.3, configured as a nonlinear transversal filter. It has been shown by Kolmogorov [21] that such a network with a single hidden layer and a sufficient number of neurons is capable of computing (with some set of weight vectors) any continuous nonlinear function of the inputs to any degree of accuracy.

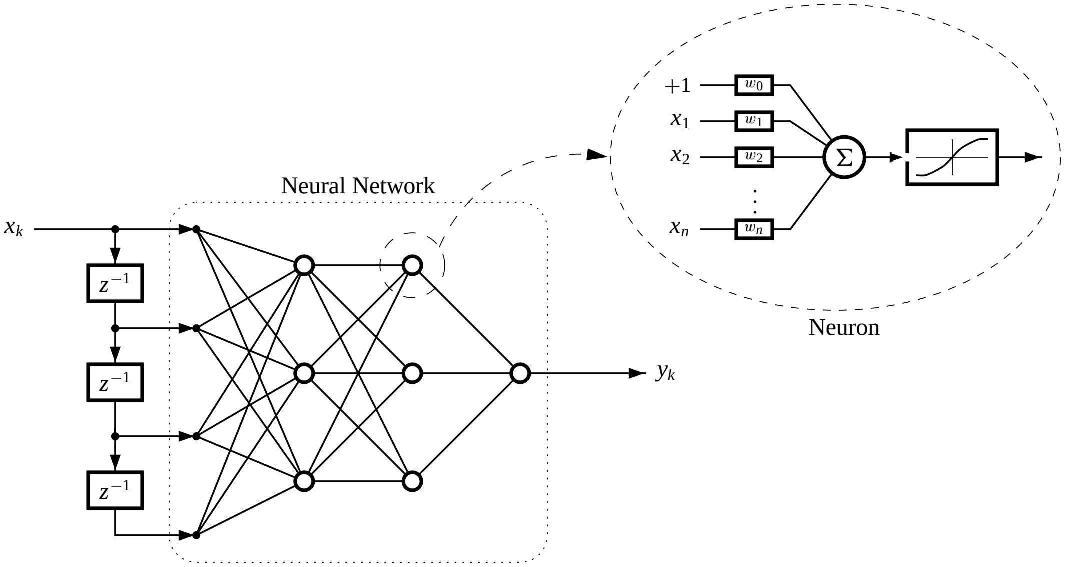


Figure A.3: Nonlinear transversal filter.

The most popular learning algorithm, due mostly to its simplicity and its robustness, is the "Error Backpropagation" algorithm. This algorithm was originally discovered by Werbos [41] but was independently developed and popularized by Rumelhart, Hinton and Williams in 1986 [35]. The backpropagation algorithm, like LMS, is a stochastic gradient descent algorithm for weight update.

To derrive the backpropagation algorithm, we consider the neurons to be numbered, such that X_i is the input vector to the ith neuron, and W_i, s_i, and y_i are the weight vector, internal sum and output of that neuron, respectively. Again, we use the mean squared error cost function, so at any point in time, k, the output cost function is

$$J_k = \frac{1}{2} e_k^2,$$

where

$$e_k = d_k - y_k.$$

For a neuron i in the output layer, the corresponding gradient is:

$$\begin{aligned}
\frac{\partial J_k}{\partial W_{i,k}} &= \frac{\partial J_k}{\partial s_{i,k}} \frac{\partial s_{i,k}}{\partial W_{i,k}} \\
&= -e_{i,k} f'(s_{i,k}) X_{i,k}.
\end{aligned}$$

To compute the gradient of the error for neurons in the hidden layers, the chain rule must be used:

$$\begin{aligned}
\frac{\partial J_k}{\partial W_{i,k}} &= \frac{\partial J_k}{\partial s_{i,k}} \frac{\partial s_{i,k}}{\partial W_{i,k}} \\
&= \frac{\partial J_k}{\partial y_{i,k}} \frac{\partial y_{i,k}}{\partial s_{i,k}} X_k \\
&= \frac{\partial J_k}{\partial y_{i,k}} f'(s_{i,k}) X_k \\
&= \left[\sum_j \frac{\partial J_k}{\partial s_{j,k}} \frac{\partial s_{j,k}}{\partial y_{i,k}} \right] f'(s_{i,k}) X_k \\
&= \left[\sum_j \delta_{j,k} w_{i:j,k} \right] f'(s_{i,k}) X_k \\
&= f'(s_{i,k}) X_k \sum_j \delta_{j,k} w_{i:j,k}.
\end{aligned}$$

The summation over j in the fourth line is the summation over all neurons to which the output y_i of neuron i is connected. $\delta_{j,k}$ then, is the derivative of squared error with respect to $s_{j,k}$ and is recursively computed backwards through the network.

To summarize, the weights of the network are changed as follows:

$$W_{i,k+1} = W_{i,k} - \mu\delta_{i,k}X_{i,k},$$

where $W_{i,k}$ is the value of the weight vector in the ith neuron after the kth learning step.

μ is the learning rate.

$X_{i,k}$ is the input to the ith neuron, including the bias input of 1.

$\delta_{i,k}$ is the derivative of the squared error for that neuron.

For an output layer neuron,

$$\delta_{i,k} = -e_{i,k}\,f'(s_{i,k}),$$

where $s_{i,k}$ is the result of the scalar product in the neuron. For any other neuron,

$$\delta_{i,k} = f'(s_{i,k})\sum_j \delta_j w_{i:j,k},$$

where $w_{i:j,k}$ is the weight connecting the output of the ith neuron to the input of the jth neuron. The summation is over all neurons fed by the output of neuron i.

A.3.1 Stability of the Backpropagation Algorithm

Stability for a Single Adaline

Before addressing the issue of stability for a neural network whose weights are adapted by the backpropagation algorithm, we first consider the simpler case of a single neuron. Such a neuron, or Adaline, is shown in Fig. A.2.

We proceed to analyze it in a similar fashion to the adaptive linear filter. The Lyapunov function is again chosen to be:

$$V_k = \frac{1}{2}e_k^2.$$

The difference in the Lyapunov function from time step k to time step $k+1$ is:

$$\Delta_{V_k} = V_{k+1} - V_k = \frac{1}{2}\left[e_{k+1}^2 - e_k^2.\right]$$

The error difference due to the learning can be represented by

$$e_{k+1} = e_k + \Delta_{e_k}.$$

Again, assuming that the input and desired responses remain the same,

$$\begin{aligned}
e_{k+1} &= d_k - y_{k+1}\\
&= d_k - f(W_{k+1}^T X_k)\\
e_k &= d_k - y_k\\
&= d_k - f(W_k^T X_k)\\
\Delta_{e_k} &= e_{k+1} - e_k\\
&= -\big(f(W_{k+1}^T X_k) - f(W_k^T X_k)\big)\\
&= -\Big(f(W_k^T X_k + \Delta_{W_k^T} X_k) - f(W_k^T X_k)\Big)\\
&\approx -\left(\left.\frac{\partial f(x)}{\partial x}\right|_{x=W_k^T X_k} \cdot \Delta_{W_k^T} X_k\right)\\
&= -f'(W_k^T X_k)\Delta_{W_k}^T X_k,
\end{aligned}$$

where $f(x)$ is the sigmoidal function used by the neuron, and the approximation is used by expanding the Taylor series and discarding all terms of second order and higher. To proceed, we define $s_k = W_k^T X_k$, and replace Δ_{W_k} by the backpropagation weight update equation:

$$\Delta e_k \approx -\mu[f'(s_k)]^2 e_k X_k^T X_k.$$

Therefore,

$$\begin{aligned}
\Delta_{V_k} &= \frac{1}{2}\left(2e_k\Delta_{e_k} + \Delta_{e_k}^2\right)\\
&= \Delta_{e_k}\left(e_k + \frac{1}{2}\Delta_{e_k}\right)\\
&= -\mu[f'(s_k)]^2 e_k \|X_k\|^2\left(e_k - \frac{1}{2}\mu[f'(s_k)]^2 e_k \|X_k\|^2\right)\\
&= -\mu[f'(s_k)]^2 e_k^2 \|X_k\|^2\left(1 - \frac{1}{2}\mu[f'(s_k)]^2 \|X_k\|^2\right).
\end{aligned}$$

For Δ_{V_k} to be negative, we need that:

$$0 < \mu < \frac{2}{[f'(s_k)]^2\|X_k\|^2}.$$

Stability for an Entire Neural Network

In order to extend the above analysis to a network of neurons, with possibly many outputs, the mathematical steps can become very complicated [4]. Happily, there is an easier solution. Rather than retaining the cost function $V_k = (1/2)e_k^2$, we use a vector cost function $V_k = (1/2)E_k^T E_k$, where the vector E has one entry for each neuron in the network, and contains the errors of each neuron. V_k, then, is equal to the sum of the squares of the errors at the output of each neuron. A sufficient condition for V_k to be decreasing is for each of the errors to be decreasing. Furthermore, we allow each neuron to have its own learning rate $\mu_{i,k}$. Therefore, for all output neurons, we have already solved the stability criteria and have discovered the acceptable range of learning rate. It only remains to determine the range of learning rate for hidden neurons.

A hidden neuron has no explicit desired response, and hence no measure of error. However, we can construct an error "by analogy" with an output layer neuron, and state:

$$e_{i,k} = \frac{-\delta_{i,k}}{f'(s_{i,k})}.$$

Therefore,

$$\begin{aligned}
\Delta_{e_{i,k}} &= \mu_i f'(s_{i,k})\delta_{i,k}\|X_{i,k}\|^2 \\
\Delta_{V_{i,k}} &= \Delta_{e_{i,k}}\left(e_{i,k} - \Delta_{e_{i,k}}\right) \\
&= \mu_i f'(s_{i,k})\|X_{i,k}\|^2\delta_{i,k}\left(\frac{-\delta_{i,k}}{f'(s_{i,k})} + \frac{1}{2}\mu_i f'(s_{i,k})\delta_{i,k}\|X_{i,k}\|^2\right) \\
&= \mu_i\delta_{i,k}^2\|X_{i,k}\|^2\left(-1 + \frac{1}{2}\mu_i\left(f'(s_{i,k})\right)^2\|X_{i,k}\|^2\right).
\end{aligned}$$

From the last line, we can conclude that the condition on μ_i for stability is the same for hidden layer neurons as for output neurons! Therefore, we conclude:

$$0 < \mu_i < \frac{2}{[f'(s_{i,k})]^2\|X_{i,k}\|^2},$$

for each neuron i in the network.

As with the linear filter, there are a number of ways to use this result. The method used in this work is to choose:

$$\mu_{i,k} = \min_{0\le j<k}\frac{1}{[f'(s_{i,k})]^2\|X_{i,k}\|^2}.$$

A.4 Summary

This appendix proposes methods to adaptively control the learning rate when using the LMS or backpropagation algorithms. The analysis is based on an approximate Lyapanov method, and attempts to find the maximum learning rate which will ensure convergence. A rigorous theoretical treatment of the convergence properties of the algorithms is difficult and has yet to appear. However, these approximate analyses give very useful insight into the stable range for each of these algorithms. In all cases, checks with computer simulation indicate that the analytical results presented here are sufficiently accurate for design purposes.

The adaptive learning rates were used with all simulations reported in this dissertation. While the results were not derived for the case of recurrent neural networks and linear filters, it was found in practice that they worked just as well when “blindly” applied even in those cases. Much frustration was saved by having the adaptive system find the right “ballpark” learning rate automatically.

Appendix B

More Nonlinear SISO Examples

Example is not the main thing in influencing others. It is the only thing.
—Albert Schweitzer

B.1 Introduction

In the main text, a few practically motivated control tasks were chosen to illustrate the principal results of this dissertation. Here, rather than burden the previous discussion, we consider seven more SISO nonlinear plants. These have been proposed by other researchers investigating adaptive inverse control for nonlinear SISO systems [2, 4]. This appendix will show that the methods developed in this dissertation are able to control these seven systems. The plants are defined, system identification is performed (in the presence and absence of disturbance), feedforward control is accomplished, and disturbance cancelling is demonstrated.

B.2 System Identification

This section describes the seven different nonlinear SISO plants, and demonstrates system identification for each of them. First, the systems are identified in the *absence* of disturbance, and a summary plot of the system identification process is presented in Fig. B.8. Secondly, system identification is performed, starting again with random weight values, in the *presence* of disturbance, and a summary plot is presented in Fig. B.9. For the cases where a recurrent plant model was used, it was trained using a series-parallel method first to accomplish coarse training. Final training was always in a parallel connection to ensure unbiased results. The RTRL algorithm was used.

A variety of plants are considered. The collection includes nonlinear FIR and nonlinear IIR plants; plants described in state-space, plants described with nonlinear difference equations (and combinations of the two); and both minimum-phase and nonminimum-phase plants. Similarly, a variety of ways to inject disturbance are considered. Since the plants are not motivated by any particular 'real' dynamical system, the command signal and disturbance sources are artificial as well. In each case, command signal is uniform i.i.d., which was chosen since it is the most difficult to follow. The raw disturbance source is a first-order Markov process. In some cases the Markov process is driven by i.i.d. uniform random variables, and in other cases by i.i.d. Gaussian random variables. The disturbance is added to either the input of the system, to the output of the system, to a specific state in the plant's state-space representation or to an intermediate stage of the processing performed by the plant.

System 1: The first plant we consider was initially proposed in [26]. The plant's block diagram is shown in Fig. B.1, and the difference equations defining its dynamics are:

$$\begin{aligned} s_k &= \frac{s_{k-1}}{1+s_{k-1}^2} + u_{k-1}^3 \\ y_k &= s_k + \mathrm{dist}_k. \end{aligned}$$

Plant identification in the absence of disturbance was performed using a $\mathcal{N}_{(2,1):8:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-2,2]$. Results of system identification are shown in Fig. B.8.

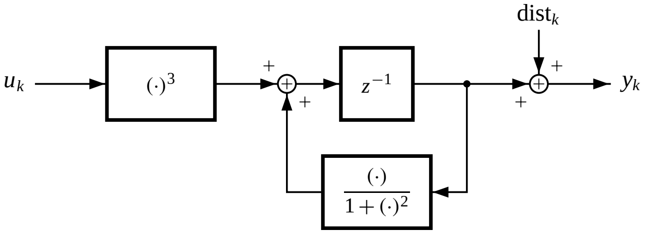


Figure B.1: Block diagram of system 1.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were uniformly distributed in the range $[-0.5, 0.5]$. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added directly to the output of the system. Note that the disturbance is added *after* the nonlinear filter, and hence it does not affect the internal state of the system.

Plant identification in the presence of disturbance was performed using the same $\mathcal{N}_{(2,1):8:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-2,2]$. Results of system identification are shown in Fig. B.9. The plant model very closely approximates the *undisturbed* dynamics of the system (which is the goal).

System 2: The second nonlinear SISO system is a generalization of the first. The plant's block diagram is shown in Fig. B.2 and the difference equations defining its plant's dynamics are:

$$\begin{aligned} s_k &= \frac{s_{k-1}}{1+s_{k-1}^2} + \sin(u_{k-1}) \\ y_k &= s_k + \text{dist}_k. \end{aligned}$$

Plant identification was performed in the absence of disturbance using a $\mathcal{N}_{(2,1):3:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1,1]$. Results of system identification are shown in Fig. B.8.

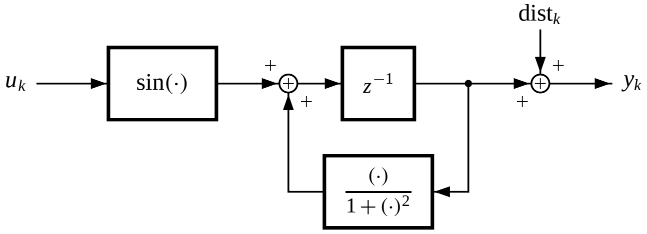


Figure B.2: Block diagram of system 2.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were distributed according to a Gaussian distribution with zero mean and standard deviation 0.1. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added directly to the output of the system, as with system 1, and does not affect the internal state of the system. Plant identification was performed in the presence of disturbance using the same $\mathcal{N}_{(2,1):3:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1,1]$. Results of system identification are shown in Fig. B.9.

System 3: The third plant is a nonlinear transversal system, defined by:

$$y_k = \tan^{-1}(u_{k-1} - 0.5u_{k-2} + \text{dist}_{k-1}).$$

Its block diagram is shown in Fig. B.3. Plant identification was performed in the absence of disturbance using a $\mathcal{N}_{(3,0):2:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-2, 2]$. Results of system identification are shown in Fig. B.8.

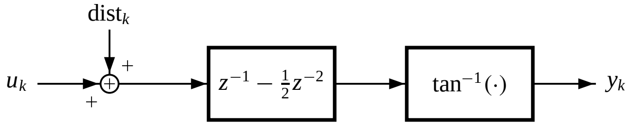


Figure B.3: Block diagram of system 3.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were uniformly distributed in the range $[-0.05, 0.05]$. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added to the input of the system. Plant identification was performed in the presence of disturbance using the same $\mathcal{N}_{(3,0):2:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-2, 2]$. Results of system identification are shown in Fig. B.9.

System 4: The fourth plant is also a nonlinear transversal system. Unlike system 3, however, it is also nonminimum-phase. The difference equation defining its dynamics is:

$$y_k = \exp(u_{k-1} - 2u_{k-2} + \mathrm{dist}_{k-1}) - 1.$$

Its block diagram is shown in Fig. B.4. Plant identification was performed using a $\mathcal{N}_{(3,0):3:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-0.5, 0.5]$. Results of system identification are shown in Fig. B.8.

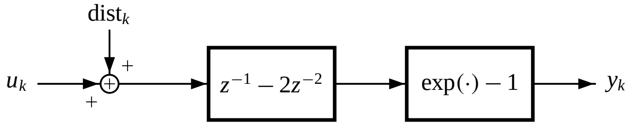


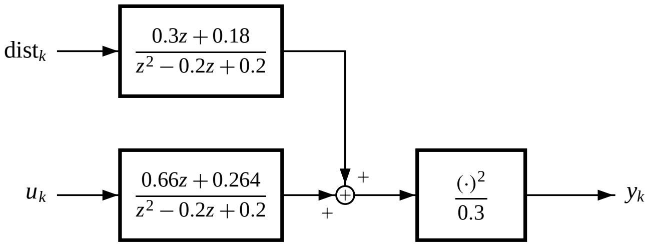
Figure B.4: Block diagram of system 4.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were distributed according to a Gaussian distribution with zero mean and standard deviation 0.03. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added to the input of the system, as with system 3. Plant identification was performed in the presence of disturbance using the same $\mathcal{N}_{(3,0):3:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-0.5, 0.5]$. Results of system identification are shown in Fig. B.9.

System 5: The fifth system is a nonlinear plant expressed in state-space form. An equivalent transfer-function form is shown in Fig. B.5. The system consists of a linear filter followed by a squaring device. The difference equations defining this plant's dynamics are:

$$\begin{aligned} x_k &= \begin{bmatrix} 0 & 1 \\ -0.2 & 0.2 \end{bmatrix} x_{k-1} + \begin{bmatrix} 0.2 \\ 1 \end{bmatrix} u_{k-1} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathrm{dist}_{k-1} \\ s_k &= \begin{bmatrix} 1 & 2 \end{bmatrix} x_k \\ y_k &= 0.3\,(s_k)^2 . \end{aligned}$$

Plant identification was performed using a $\mathcal{N}_{(10,5):8:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1, 1]$. Results of system identification are shown in Fig. B.8.



dist_k
$\frac{0.3z+0.18}{z^2-0.2z+0.2}$
u_k
$\frac{0.66z+0.264}{z^2-0.2z+0.2}$
+
+
$\frac{(\cdot)^2}{0.3}$
y_k

Figure B.5: Block diagram of system 5.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were uniformly distributed in the range $[-0.5, 0.5]$. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added directly to the first state of the system. Plant identification was performed in the presence of disturbance using the same $\mathcal{N}_{(10,5):8:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1, 1]$. Results of system identification are shown in Fig. B.9.

System 6: The sixth system was first suggested in reference [45]. It comprises a linear filter followed by a nonlinear squashing function, and followed in turn by another linear filter. Its block diagram is shown in Fig. B.6. The difference equations defining this plant's dynamics are:

$$\begin{aligned} s_k &= 0.4 s_{k-1} + 0.5 u_k \\ y_k &= 0.8 y_{k-1} + \tanh(s_k) + \mathrm{dist}_k . \end{aligned}$$

Plant identification was performed using a $\mathcal{N}_{(5,1):10:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1,1]$. Results of system identification are shown in Fig. B.8.

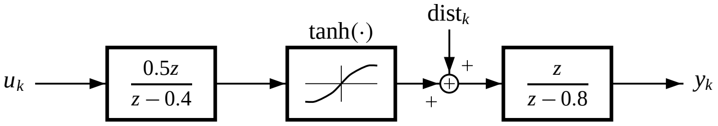


Figure B.6: Block diagram of system 6.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were uniformly distributed in the range $[-0.05, 0.05]$. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.95$. The resulting disturbance was added to an intermediate point in the system, just before the output filter. Note that the disturbance affects the state y_k of the system. Plant identification was performed in the presence of disturbance using the same $\mathcal{N}_{(5,1):10:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1,1]$. Results of system identification are shown in Fig. B.9.

System 7: The final system is a generalization of the sixth system. The nonlinearity in the sixth system is static—it has no memory. In the seventh system, this static nonlinearity is replaced by one having memory or "phase." It is a type of hysteresis device. This device makes the system have two equilibrium states as opposed to the previous plants which all had a single equilibrium state. The system is shown in Fig. B.7 and the difference equations defining its dynamics are:

$$\begin{aligned} s_k &= 0.4s_{k-1} + 0.5u_k \\ y_k &= \text{dist}_k + \begin{cases} 0.8y_{k-1} + 0.8\tanh(s_k - 2), & \text{if } s_k > s_{k-1}; \\ 0.8y_{k-1} + 0.8\tanh(s_k + 2), & \text{if } s_k \le s_{k-1}. \end{cases} \end{aligned}$$

Plant identification was performed using a $\mathcal{N}_{(10,10):30:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1,1]$. Results of system identification are shown in Fig. B.8.

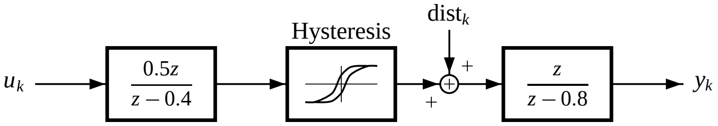


Figure B.7: Block diagram of system 7.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were distributed according to a Gaussian distribution with zero mean and standard deviation 0.01. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.95$. The resulting disturbance was added to an intermediate point in the system, just before the output filter, as with system 6. Plant identification was performed in the presence of disturbance using the same $\mathcal{N}_{(10,10):30:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1, 1]$. Results of system identification are shown in Fig. B.9.

Summary of System Identification: In all cases, a neural network was found which satisfactorily identified the system. Each system was driven with an i.i.d. uniform control signal. This was not characteristic of the control signal generated by the trained controller in the next section, but was a starting point and worked quite well to initialize the plant model for use in training the controller. Each plant produced its own very characteristic output for the same input, as seen in Fig. B.8, but it was shown that neural networks could be trained to identify each system nearly perfectly.

When disturbance is added, it is useful to think of the "disturbed plant dynamics" and the "nominal plant dynamics." In each case, the system identification process matched the nominal dynamics of the plant, which is what theory predicts, and what we would like.

B.3 Feedforward Control

After system identification was done, the controller was trained to perform feedforward control of each system. The control input signal was always uniform i.i.d. random input. Since the plants themselves are artificial, this artificial control signal was chosen. In general, it is the hardest control signal to follow. The plants were undisturbed. Disturbance cancelling for disturbed plants is considered in the next section.

Note that the control signal generated by the trained controller is not an i.i.d. uniform signal. Therefore, the system identification performed in the previous section is not sufficient to properly train the controller. It provides a very good initial set of values for the weights of the controller, however, and system identification continues on-line as the controller is trained with the BPTM algorithm.

First, the controller was trained with an i.i.d. uniform command input. The reference model, in all cases (except for system 4), was a unit delay. When the weights had converged, the values of the network weights were frozen, and the controller was tested with an i.i.d. uniform, a sinusoidal and

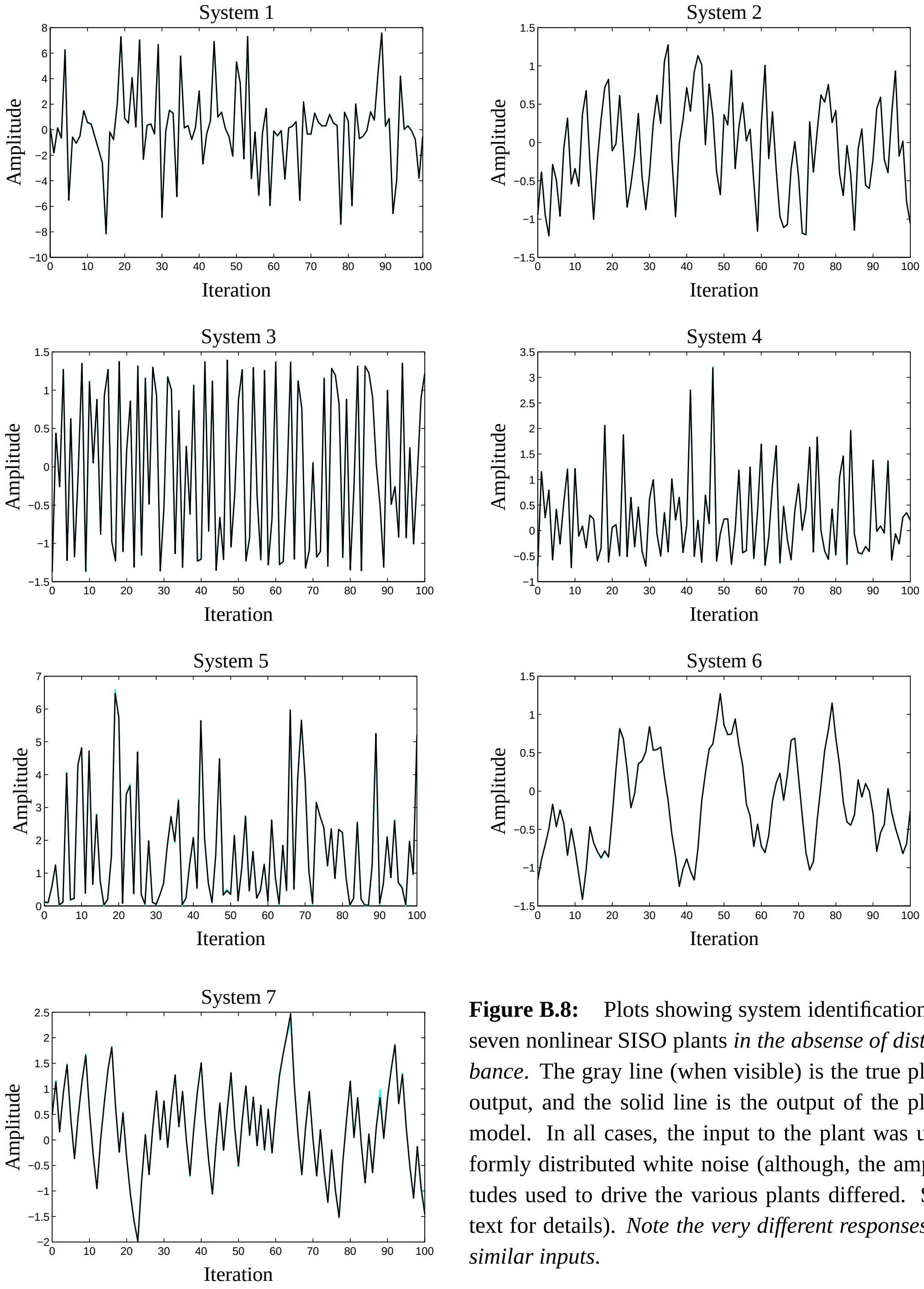


Figure B.8: Plots showing system identification of seven nonlinear SISO plants *in the absense of disturbance*. The gray line (when visible) is the true plant output, and the solid line is the output of the plant model. In all cases, the input to the plant was uniformly distributed white noise (although, the amplitudes used to drive the various plants differed. See text for details). *Note the very different responses to similar inputs.*

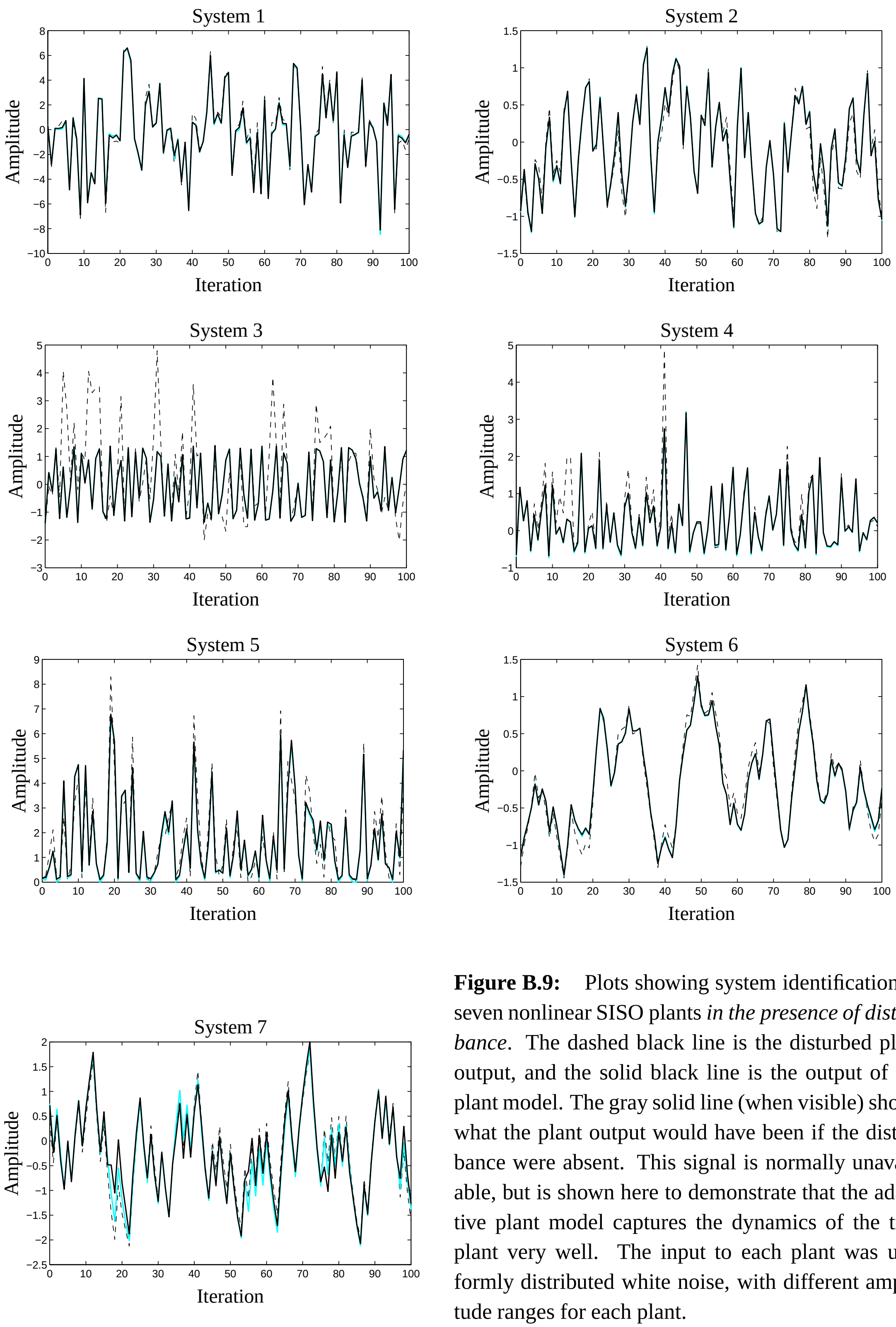


Figure B.9: Plots showing system identification of seven nonlinear SISO plants *in the presence of disturbance*. The dashed black line is the disturbed plant output, and the solid black line is the output of the plant model. The gray solid line (when visible) shows what the plant output would have been if the disturbance were absent. This signal is normally unavailable, but is shown here to demonstrate that the adaptive plant model captures the dynamics of the true plant very well. The input to each plant was uniformly distributed white noise, with different amplitude ranges for each plant.

a square wave to show the performance and generalization of the system. The results are presented in Figs. B.11 to B.17.

Generally (specific variations will be addressed below) the tracking of the uniform i.i.d. signal was nearly perfect, and the tracking of the sinusoidal and square waves was excellent as well. We see that the control signals generated by the controller are quite different for the different desired trajectories, so the controller can generalize well. When tracking a sinusoid, the control signal for a linear plant is also sinusoidal. Here, the control signal is never sinusoidal, indicating in a way the degree of nonlinearity in the plants.

Notes on System 4: System 4 is a nonminimum-phase plant. This can be easily verified by noticing that its linear-filter part has a zero at $z = 2$. This plant cannot follow a unit-delay reference model. Therefore, reference models of different delays were tried, for delays of zero time samples up to 15 time samples. In each case, the controller was fully trained, and the steady-state mean-squared-system error was measured. A plot of the results is shown in Fig. B.10. Since both low MSE and low delay is desirable, a reference model for this work was chosen to be a delay of ten time samples: $M(z) = z^{-10}$.

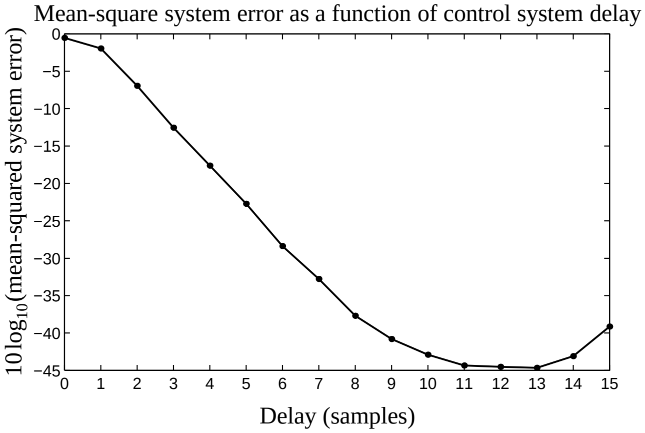
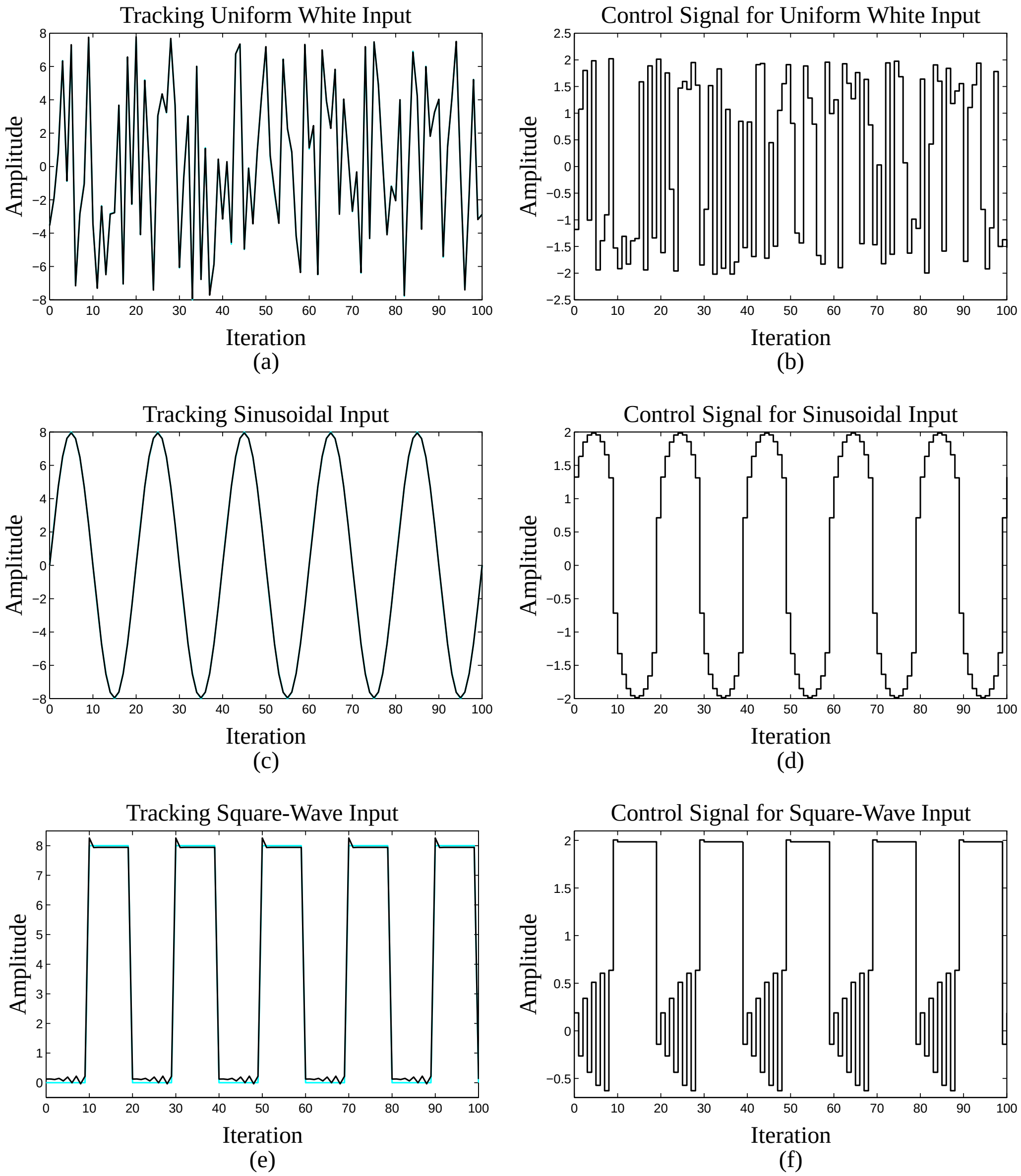


Figure B.10: Logarithm of mean-squared system error plotted versus control system delay.

Notes on System 5: The output of system 5 is constrained to be positive due to the squaring device in its representation. Therefore it is interesting to see how well this system generalizes when asked to track a zero-mean sinusoidal signal. As shown in Fig. B.15, the result is something like a full-wave rectified version of the desired result. This is neither good nor bad—just curious.



(a) (b) (c) (d) (e) (f)

Figure B.11: Feedforward control of system 1. The controller, $\mathcal{N}_{(2,1):6:1}$, was trained to track uniformly distributed (white) random input, between [-8,8]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals.

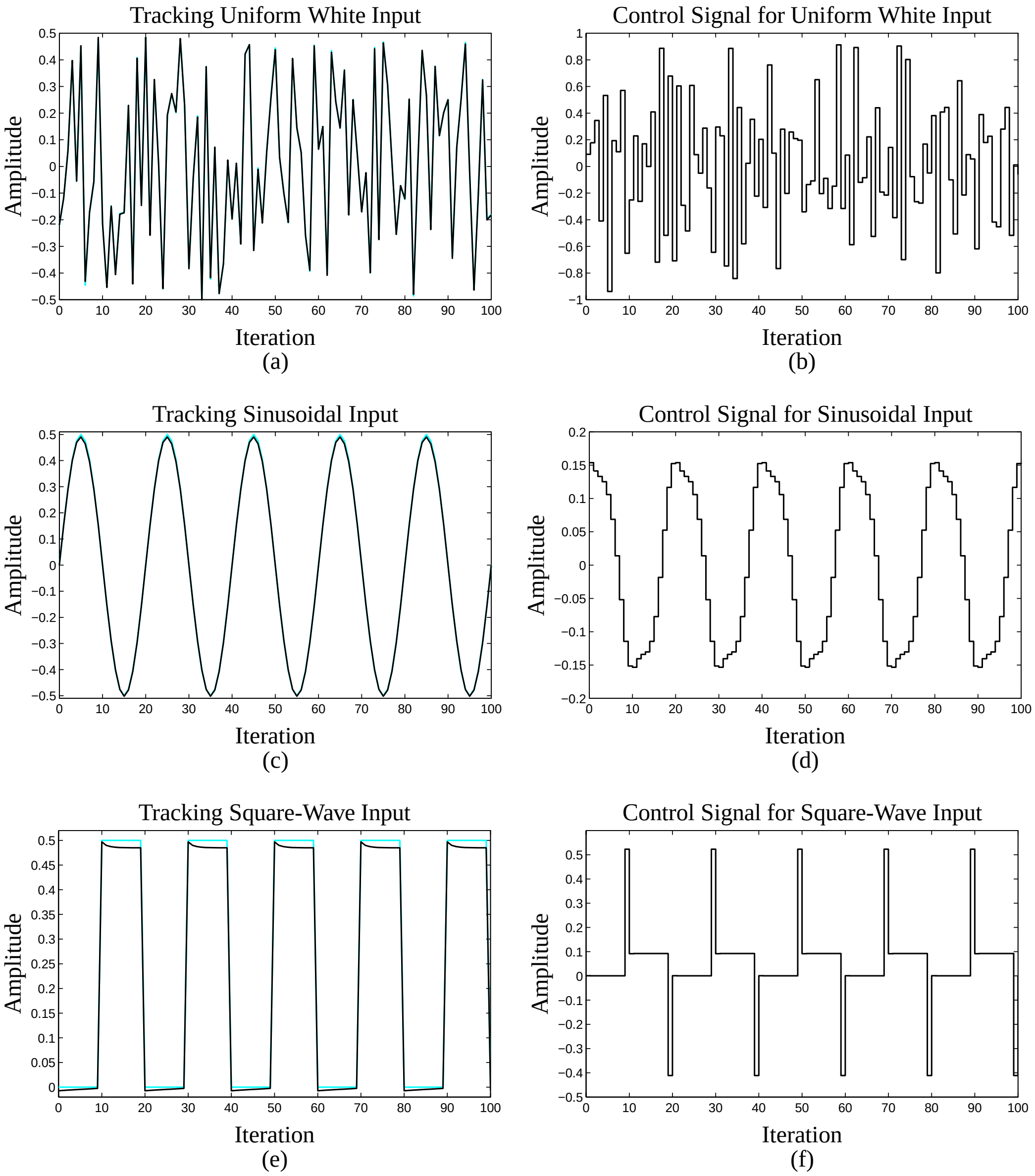


Figure B.12: Feedforward control of system 2. The controller, $\mathcal{N}_{(2,1):10:1}$, was trained to track uniformly distributed (white) random input, between [-0.5,0.5]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals.

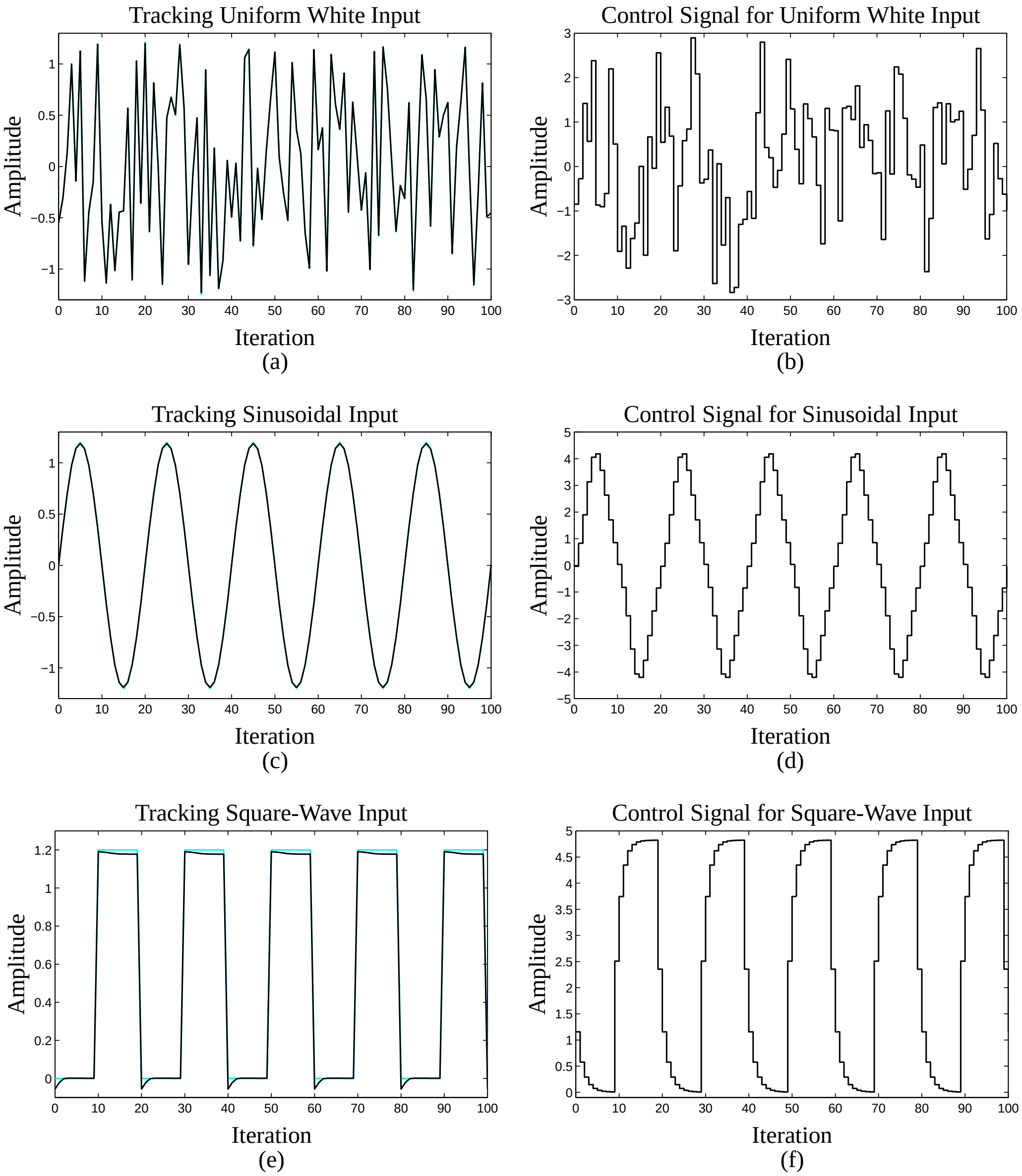
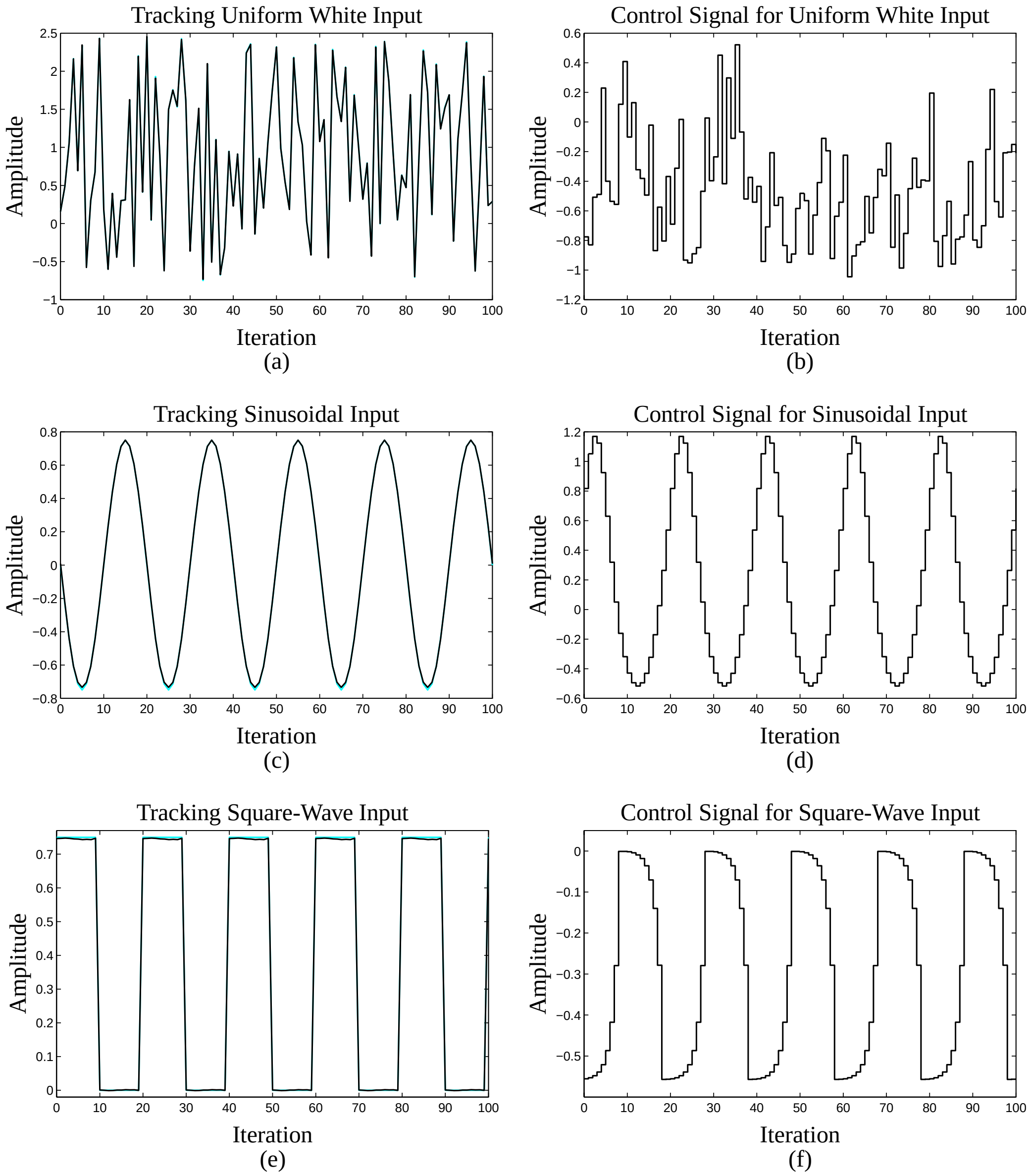


Figure B.13: Feedforward control of system 3. The controller, $\mathcal{N}_{(2,1):20:1}$, was trained to track uniformly distributed (white) random input, between [-1.25,1.25]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals.

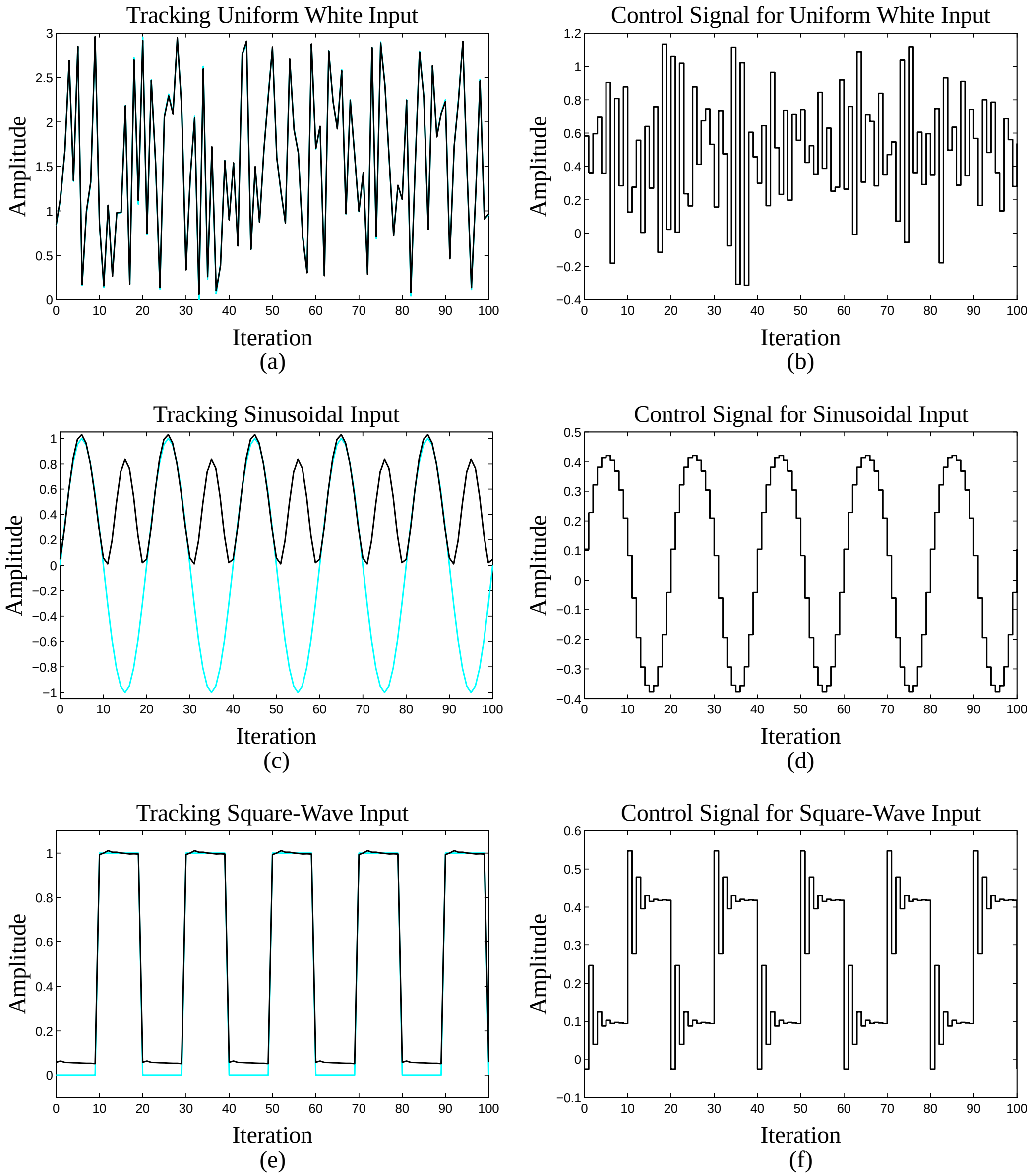


(a) (b)

(c) (d)

(e) (f)

Figure B.14: Feedforward control of system 4. The controller, Net(20,1):20:1, was trained to track uniformly distributed (white) random input, between [-0.75,2.5]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals.



(a) (b) (c) (d) (e) (f)

Figure B.15: Feedforward control of system 5. The controller, $\mathcal{N}_{(2,8):8:1}$, was trained to track uniformly distributed (white) random input, between [0,3]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals.

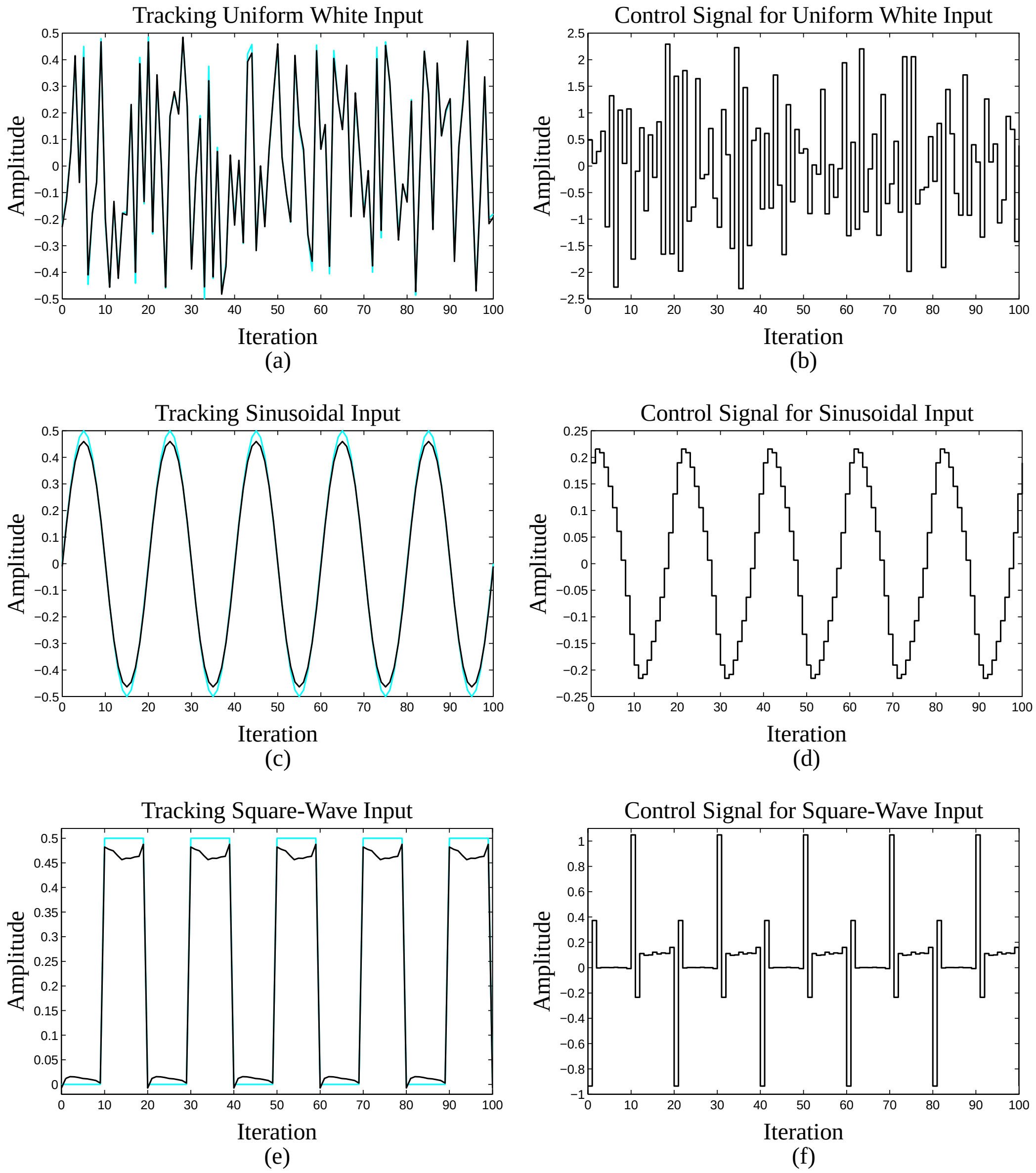
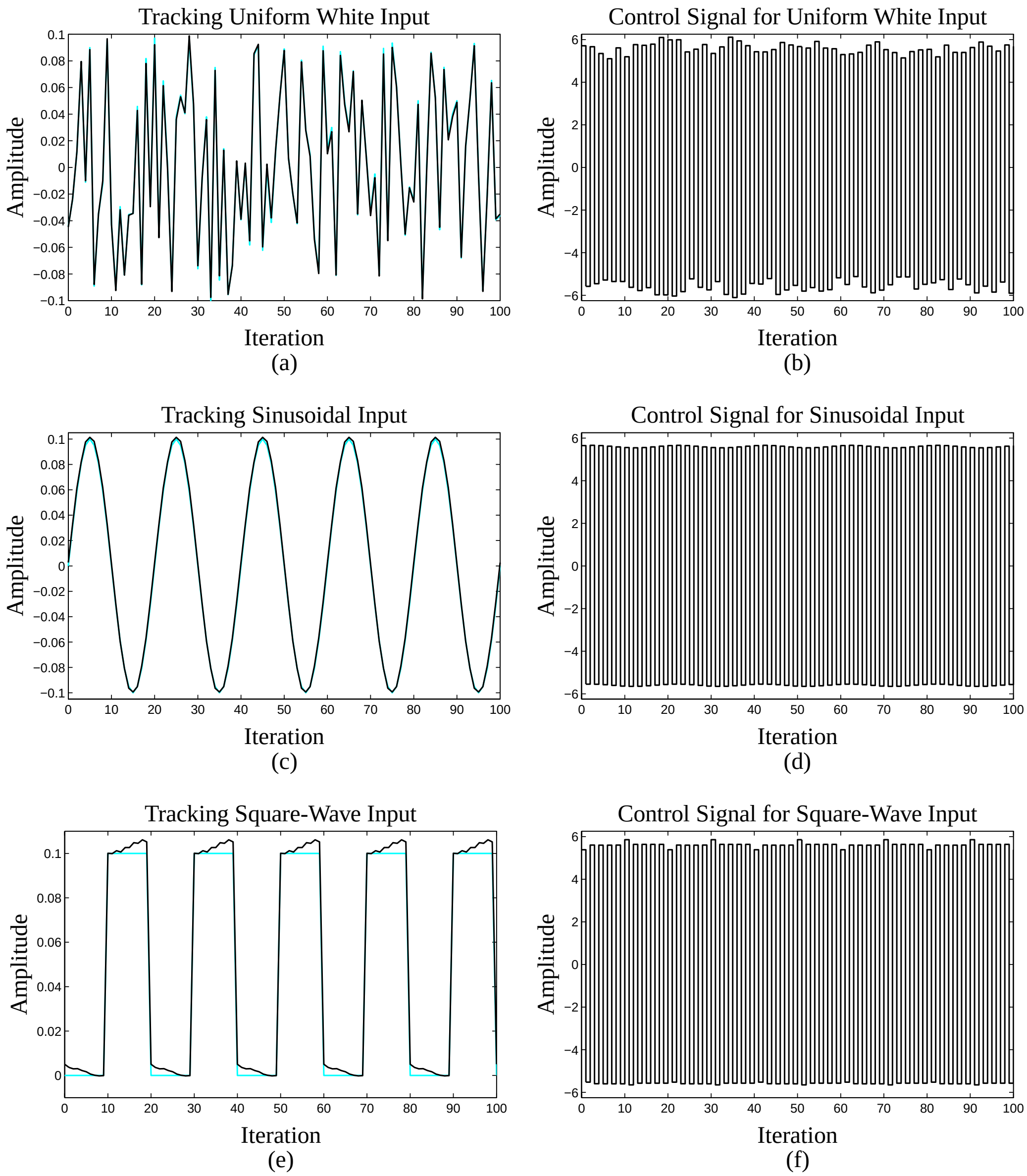


Figure B.16: Feedforward control of system 6. The controller, $\mathcal{N}_{(10,1):10:1}$, was trained to track uniformly distributed (white) random input, between [-0.5,0.5]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals.



(a) (b) (c) (d) (e) (f)

Figure B.17: Feedforward control of system 7. The controller, $\mathcal{N}_{(10,1):30:1}$, was trained to track uniformly distributed (white) random input, between [-0.1,0.1]. Plot (a) depicts the desired plant output (gray line) and the true plant output (solid line), at the end of training, when the training signal was used to drive the controller. Plot (b) shows the controller output for this case. With the weights fixed at their trained values, the next four plots show the generalization ability of the controller. Plots (c) and (e) show the plant tracking a sinusoidal and square wave, and plots (d) and (f) are the corresponding plant input signals. *Notice that the control signals are almost identical for these three very different inputs!*

Simulations were also done to see what would happen if the system were *trained* to follow this kind of command input. In that case, the plant output looks like a half-wave rectified version of the input. Indeed, that is the result which minimizes MSE—the training algorithm works!

Notes on System 7: As can be seen from Fig. B.17, the control signal required to control this plant is extremely harsh. The hysteresis in the plant requires a type of modulated bang-bang control. Notice that the control signals for the three different inputs are almost identical. The plant is very sensitive to its input, yet can be controlled very well by a neural network trained with the BPTM algorithm.

B.4 Disturbance Cancelling

With system identification and feedforward control accomplished, disturbance cancellation was performed. The input to the disturbance cancelling filter X was chosen to be tap-delayed copies of the u_k and $\widehat{w}_k$ signals. The architecture of the filter chosen for each system is listed in Table. B.1.

The BPTM algorithm was used to train the disturbance cancellers. After training, the performance of the cancellers was tested and the results are shown in Fig. B.18. In this figure, each system was run with the disturbance canceller turned off for 500 time samples, and then turned on for the next 500 time samples. The squared system error is plotted. The disturbance cancellers do a fantastic job of removing the disturbance from the systems.

TABLE B.1 FILTER ARCHITECTURES USED IN THIS APPENDIX.

System	Plant Model, $\widehat{P}$	Feedforward Controller, C	Disturbance Canceler, X
1	$\mathcal{N}_{(2,1):8:1}$	$\mathcal{N}_{(2,1):6:1}$	$\mathcal{N}_{([5,5],2):10:1}$
2	$\mathcal{N}_{(2,1):3:1}$	$\mathcal{N}_{(2,1):10:1}$	$\mathcal{N}_{([4,3],4):10:1}$
3	$\mathcal{N}_{(3,0):2:1}$	$\mathcal{N}_{(2,1):20:1}$	$\mathcal{N}_{([5,5],1):10:1}$
4	$\mathcal{N}_{(3,0):3:1}$	$\mathcal{N}_{(20,1):20:1}$	$\mathcal{N}_{([5,5],1):10:1}$
5	$\mathcal{N}_{(10,5):8:1}$	$\mathcal{N}_{(2,8):8:1}$	$\mathcal{N}_{([5,5],4):10:1}$
6	$\mathcal{N}_{(5,1):10:1}$	$\mathcal{N}_{(10,1):10:1}$	$\mathcal{N}_{([20,10],2):20:1}$
7	$\mathcal{N}_{(10,10):30:1}$	$\mathcal{N}_{(10,1):30:1}$	$\mathcal{N}_{([5,5],1):30:1}$

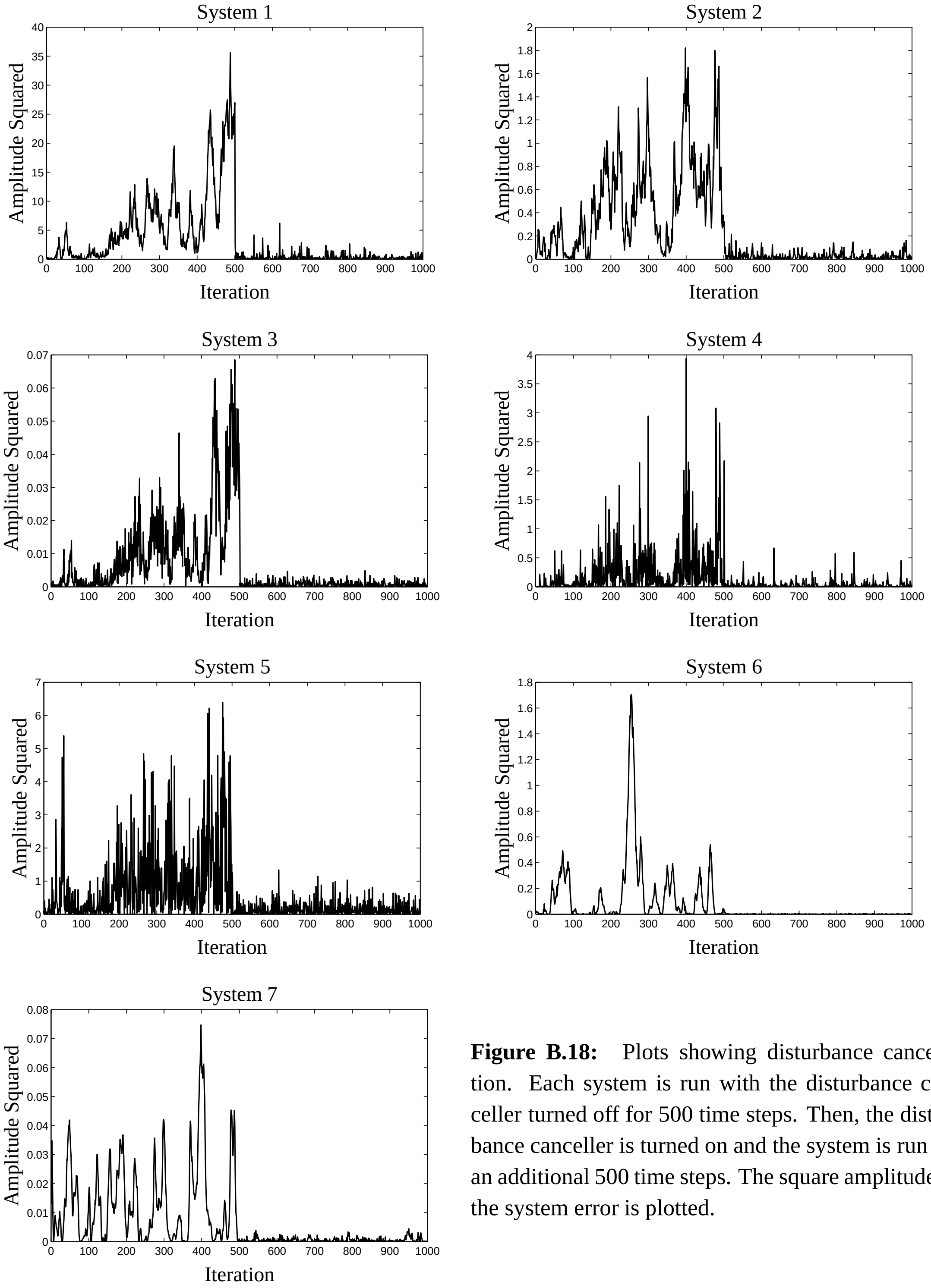


Figure B.18: Plots showing disturbance cancelation. Each system is run with the disturbance canceller turned off for 500 time steps. Then, the disturbance canceller is turned on and the system is run for an additional 500 time steps. The square amplitude of the system error is plotted.

B.5 Summary

This appendix furnishes several more examples of adaptive inverse control applied to nonlinear SISO plants. In total, seven new plants were defined. First, adaptive system identification (in the absence and presence of disturbance) was performed using a white driving signal u_k. In all cases, a neural network plant model was able to capture the dynamics of the system very well.

Secondly, feedforward controllers C were trained. The desire was to train a controller such that the output of the system would track an i.i.d. uniform random process. This is an unrealistically difficult goal; yet, the neural-network controllers learned to do it with very high precision. It should be mentioned that no previous success at controlling one of the systems (system 7) has been reported—this is the first time that it has been achieved.

The output of the trained controller is certainly not similar to the white driving signal used to train the plant model, so simultaneous adaptation of the controller and plant model are necessary. This worked well, as long as the plant model was allowed to adapt somewhat more quickly than the controller. The ability of the plant model to generalize also ensured that the initial weight values yeilded a good model regardless of the input signal.

Training of the controllers was temporarily stopped and the weight values were frozen. The system was then tested to see how well it would track input signals which were very different from those used when training the controllers. Surprisingly good tracking was achieved, showing that the neural-network controllers generalize well.

Finally, disturbance cancelling filters X were adapted. The disturbance sources corrupted either the input, output, or internal state of each system. In all cases, the disturbance canceller adapted to eliminate almost all of the disturbance. The overall conclusion of this appendix is that the methods presented in this dissertation control these seven plants very well.

Bibliography

When I get a little money I buy books. And if there is any left over, I buy food.
—Erasmus

[1] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, Reading, MA, second edition, 1995.

[2] M. Bilello. *Nonlinear Adaptive Inverse Control.* PhD thesis, Stanford University, Stanford, CA, April 1996.

[3] S. P. Boyd. *Linear controller design: Limits of performance.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[4] D. Carbonell Oliver. *Neural Networks Based Nonlinear Adaptive Inverse Control Algorithms*. Thesis for the Engineer degree, Stanford University, Stanford, CA, September 1996.

[5] C. G. Economou and M. Morari. Internal model control. 5. Extension to nonlinear systems. *Industrial and Engineering Chemistry Process Design and Development*, 25(2):403–11, April 1986.

[6] C. G. Economou and M. Morari. Internal model control. 6. Multiloop design. *Industrial and Engineering Chemistry Process Design and Development*, 25(2):411–19, April 1986.

[7] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Addison-Wesley, Reading, MA, third edition, 1994.

[8] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, Reading, MA, second edition, 1990.

[9] C. E. Garcia and M. Morari. Internal model control. 1. A unifying review and some new results. *Industrial and Engineering Chemistry Process Design and Development*, 21(2):308–23, April 1982.

[10] C. E. Garcia and M. Morari. Internal model control. 2. Design procedure for multivariable systems. *Industrial and Engineering Chemistry Process Design and Development*, 24(2):472–84, April 1985.

[11] C. E. Garcia and M. Morari. Internal model control. 3. Multivariable control law computation and tuning guidelines. *Industrial and Engineering Chemistry Process Design and Development*, 24(2):484–94, April 1985.

[12] R. Gazit. Neural control of a multi-link robot arm. Project report for Stanford University class “EE373A,B”, June 1994.

[13] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, MA, 1992.

[14] A. Grace, A. J. Laub, J. N. Little, and C. M. Thompson. *Control System Toolbox for use with MATLAB*. The Math Works Inc, Natick, MA, 1992.

[15] M. Green. *Linear Robust Control*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[16] B. Hassibi, A. H. Sayed, and T. Kailath. H_∞ optimality of the LMS algorithm. *IEEE Transactions on Signal Processing*, 44(2):267–80, February 1996.

[17] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, Upper Saddle River, NJ, third edition, 1996.

[18] R. K. Heffley and W. F. Jewell. Aircraft handling qualities. Technical Report 1004-1, System Technology, Inc., Hawthorne, CA, May 1972.

[19] T. Kailath. *Linear Systems*. Prentice Hall, Englewood Cliffs, NJ, 1980.

[20] F. C. Kaminsky, R. H. Kirchhoff, C. Y. Syu, and J. F. Manwell. A comparison of alternative approaches for the synthetic generation of a wind speed time series. *Transactions of the American Society of Mechanical Engineers. Journal of Solar Energy Engineering*, 113(4):280–89, November 1991.

[21] A. N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk USSR*, 114:953–56, 1957. (in Russian).

[22] A. U. Levin and K. S. Narendra. Control of nonlinear dynamical systems using neural networks: Controllability and stabilization. *IEEE Transactions on Neural Networks*, 4(2):192–206, 1993.

[23] M. C. Liu. *Statistical Analysis of Quantization—Extended from Widrow's Quantization Theory*. PhD thesis, Stanford University, Stanford, CA, May 1998.

[24] L. Ljung. *System Identification: Theorey for the user*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[25] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, 1994.

[26] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[27] D. Nguyen. *Applications of Neural Networks in Adaptive Control*. PhD thesis, Stanford University, Stanford, CA, June 1991.

[28] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[29] D. B. Parker. Learning logic. Technical Report Invention Report S81–64, File 1, Office of Technology Licencing, Stanford University, October 1982.

[30] M. J. Paulsen and O. Egeland. An output feedback tracking controller for ships with nonlinear damping terms. *Modeling, Identification and Control*, 17(2):97–106, 1996.

[31] S. W. Piché. Steepest descent algorithms for neural network controllers and filters. *IEEE Transactions on Neural Networks*, 5(2):198–212, March 1994.

[32] G. V. Puskorius and L. A. Feldkamp. Decoupled extended Kalman filter training of feedforward layered networks. In *Proceedings of the 1991 International Joint Conference on Neural Networks* (San Diego: 1990), volume II, pages 133–141, New York, 1991. IEEE Neural Networks Society.

[33] G. V. Puskorius and L. A. Feldkamp. Recurrent network training with the decoupled extended Kalman filter algorithm. In *Science of Artificial Neural Networks* (Orlando Florida: 21–24 April 1992), volume 1710, part 2, pages 461–473, New York, 1992. SPIE Proceedings Series.

[34] D. E. Rivera, M. Morari, and S. Skogestad. Internal model control. 4. PID controller design. *Industrial and Engineering Chemistry Process Design and Development*, 25(1):252–65, January 1986.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 8. The MIT Press, Cambridge, MA, 1986.

[36] H. T. Siegelmann, B. B. Horne, and C. L. Giles. Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics*, 27(2):208–215, April 1997.

[37] S. Singhal and L. Wu. Training multilayer perceptrons with the extended kalman algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I* (Denver: 1988), pages 133–140, San Mateo, CA, 1989. Morgan Kaufmann.

[38] J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[39] R. Sutton and I. M. Jess. A design study of a self-organizing fuzzy autopilot for ship control. *Proceedings of the Institution of Mechanical Engineers. Part I: Journal of Systems and Control Engineering*, 205:35–47, 1991.

[40] P. van Overschee and B. DeMoor. *Subspace Identification for Linear Systems*. Kluwer Academic Press, Boston, MA, 1996.

[41] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, August 1974.

[42] P. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[43] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–42, September 1990.

[44] B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[45] B. Widrow and E. Walach. *Adaptive Inverse Control*. Prentice Hall P T R, Upper Saddle River, NJ, 1996.

[46] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.

[47] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice Hall, Englewood Cliffs, NJ, 1996.